[illegible]
87 05 05/P

I

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 549 855 A2**

(12) **EUROPEAN PATENT APPLICATION**

(21) Application number: 92115676.6

(51) Int. Cl.5: **A61F 2/64**, A61F 2/68

(22) Date of filing: 12.09.92

(30) Priority: 05.12.91 US 804264

(43) Date of publication of application:
07.07.93 Bulletin 93/27

(84) Designated Contracting States:
AT BE CH DE DK ES FR GB GR IE IT LI LU NL PT SE

(71) Applicant: Otto Bock Orthopädische Industrie
Besitz- und
Verwaltungs-Kommanditgesellschaft
Industriestrasse
W-3408 Duderstadt 1(DE)

(72) Inventor: James, Kelvin B.
7955 - 98 Avenue
Edmonton, Alberta T6A 0B5(CA)

(74) Representative: Gramm, Werner, Prof.
Dipl.-Ing. et al
Patentanwälte Gramm + Lins
Theodor-Heuss-Strasse 1
W-3300 Braunschweig (DE)

(54) **System for controlling artificial knee joint action in an above knee prosthesis.**

(57) This invention relates to an above knee prothesis which employs a hydraulic damper to passively regulate the angular velocity or rotation of the artificial knee joint. A programmed microprocessor recognizes common gait patterns from information received from strain and knee angle sensors on the prosthesis. The microprocessor reacts at various transition points in the gait by activating a motor which in turn adjusts a valve assembly in the damper. The valve assembly is capable of variably and separately damping the knee joint motion in each of flexion and extension at the same time. Gait is improved because of the improved extent of control of knee action. In addition, distinct routines such as stair descending and sitting down can also be practised.

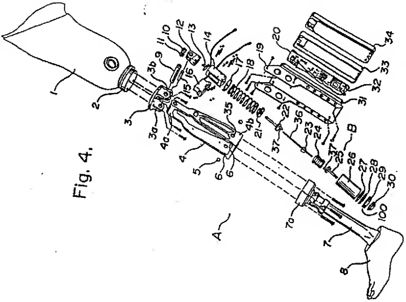


Fig. 4.

Rank Xerox (UK) Business Services

FIELD OF THE INVENTION

This invention provides a system for controlling the rotation of a knee joint of an above knee prosthesis. The system employs a microprocessor, responsive to lower leg strain and knee angle measurements originating from sensors on the prosthesis, to control a hydraulic damper through operation of a valve assembly associated with the damper, to thereby passively damp or resist the rotation of the artificial knee joint.

BACKGROUND OF THE INVENTION

As previously stated, the present invention is used with an artificial leg or prosthesis worn by an above knee amputee.

There are today about 50 different above knee prosthetic devices on the market. Many of these prostheses involve:

- a socket for receiving and engaging the stump of the user;
- a knee bracket rigidly connected to the socket;
- a frame extending down from the bracket and being pivotally connected to the bracket by a horizontal shaft, said bracket, shaft and frame together combining to form an artificial knee joint;
- a pylon and artificial foot connected to the base of the frame; and
- means for controlling the knee joint by locking it to prevent it from buckling under load in the stance phase of a step, and freeing it in the swing phase of the step.

Now, the biological or natural knee joint is powered by the actions of muscles. Muscle has two elements. One is the active force developed by contraction and the other is variable stiffness. It has not been feasible to duplicate muscle contraction in leg prosthetics, due to limitations arising from weight and bulk. As a result, research has focused on implementing stiffness into the knee joint. This has usually involved switching the knee joint between one of two modes: locked up or free to rotate.

In recent years, researchers have sought improvement in controlling the action of the artificial knee joint, as a way to improve gait and enable the amputee to better deal with certain distinct actions, such as descending stairs or lowering into a sitting position.

A relevant patent in this regard is French patent 2623-086-A. This patent teaches providing a strain gage sensor on the frame between the knee joint and foot, to measure load. The electronic signals from the sensor are transmitted to a microprocessor which monitors the load measurement. When the load signal indicates that the swing phase of the step is ending and load is being applied to the leg, the microprocessor causes a motor or electromagnet to lock up the knee joint. When the stance phase is complete, the microprocessor instructs the actuator to release the knee joint, so that it is free to pivot in the swing phase.

Another relevant prior art reference is Russian patent SU1333-333-A. This patent teaches using a sensor at the knee hinge, to measure knee angle. Means lock or free the knee hinge in response to the knee angle measurements.

Another relevant prior art device is known as the Henschke Mauch S-N-S system for controlling an above knee prosthesis. This system incorporates a linear hydraulic damper for resisting rotation of the knee joint at a single damping rate in the stance phase. The damping rate can be varied by manual adjustment. When the knee joint is fully extended, the damper assumes a non-resisting mode. Otherwise stated, the system lacks automatic variation of damping and incorporates only two states, namely high resistance to flexion in stance phase and free rotation in swing phase.

If a knee joint is looked at as a simple hinge, there are two separate actions which can occur. In "flexion", the knee joint rotates to enable the upper and lower leg segments to move closer together. In "extension" the knee joint rotates in the opposite direction, the leg segments move apart and the leg straightens. For an artificial knee joint to more closely simulate a biological knee joint, it is necessary that control or stiffness be applicable separately and variably in each of the flexion and extension modes. For example, it is desirable at the beginning of the stance (i.e. weight bearing) phase of the step to allow a small amount of knee flexion to occur and to then lock the knee against further downward flexion while simultaneously freeing the knee to extend as the leg straightens due to body action. So in the latter phase of this action, the knee joint is altered to being locked or stiff in flexion and free in extension, at the same time.

To applicant's knowledge, there is no artificial knee joint mechanism disclosed in the prior art which enables separate, simultaneous and automatic variable control of flexion and extension.

If such a mechanism could be devised, then a much more sophisticated control over the knee joint action could be implemented.

It is the object of the present invention to supply such a mechanism and to then incorporate it in an improved overall prosthesis.

SUMMARY OF THE INVENTION

The present invention relates to an on-board, computer-directed system adapted to provide improved automatic control of knee joint rotation in an above knee prosthesis (AKP) having upper and lower leg segments joined by the knee joint, said lower leg segment having a foot. In general, the system comprises:

- a linear, hydraulic damper which can separately and variably damping or resisting each of flexion and extension rotational movements of the knee joint;
- electronic sensing means for measuring each of AKP knee angle and lower leg strain (which are respectively indicative of the angle between the leg segments and the position of the center of gravity of the user's body relative to the AKP foot) and emitting signals indicative thereof;
- actuating means, such as a servo motor, for adjusting the damping means to vary the resistance to rotation of the knee joint in at least one of flexion and extension; and
- programmed computer means for receiving the emitted signals from the sensing means continuously establishing from said signals the state of the AKP in the course of a repetitive movement and activating the actuating means as required to vary damping to substantially simulate knee action. More particularly, the computer means is preferably adapted to do this by comparing the signals to stared threshold values which are indicative of pre-determined transition point between states of the AKP in the course of a movement, and, when the received signal values correlate with stored values, then causing the actuating means to vary damper resistance as required so that the AKP knee joint action substantially simulates natural knee action.

It will be noted that the invention involves separate variation of damping of AKP knee joint action in each of flexion and extension. "Damping" for this specification means resisting rotational movement of the knee joint. The resistance may be substantially complete, in which case the knee joint is substantially prevented from rotating in one or both of flexion and extension. The resistance may be partial, in which case the rate of rotation of the knee joint is restricted in one or both of flexion and extension. Or the resistance may be non-existent, in which case the knee joint is free to rotate in one or both of flexion and extension. Alternatively stated, the damper is adapted to control the rate of rotation of the knee joint in one or both of flexion and extension.

To enable such bi-directional damping, applicant has developed a novel damper incorporating a piston and means for controlling the piston. More particularly, the variable, linear, hydraulic damper comprises:

- a hollow closed cylinder filled with hydraulic fluid and having a cylindrical hollow piston adapted to slide longitudinally within the cylinder chamber;
- the piston preferably has axial rods extending from its ends, which rods project through sealed openings in the end walls of the cylinder. The piston further carries an exterior circumferential seal ring between its ends, for sealing against the side wall of the cylinder;
- a first aperture and check valve assembly, associated with a first end wall of the piston, enables fluid to enter the piston chamber from the first end of the cylinder chamber;
- a second aperture and check valve assembly, associated with the second end wall of the piston, allows fluid to enter the piston chamber from the second end of the cylinder chamber;
- a first pair of diametrically opposed ports extend through the piston side wall adjacent its first end, on one side of the seal ring;
- a second pair of diametrically opposed ports extend through the piston side wall adjacent its second end, on the other side of the seal ring;
- preferably, each first port is offset circumferentially from the second port on that side of the piston;
- preferably, each port is slit-like in configuration;
- a valve preferably extends into the cylinder and piston chambers and is adapted to progressively reduce or increase the effective area of the first (or flexion) ports available for fluid flow and separately progressively reduce or increase the area of the second (or extension) ports;
- most preferably the valve comprises a rotatable shaft extending into the piston chamber in parallel relation to the cylinder axis, said shaft carrying a pair of radially protruding, diametrically opposed lobes, each lobe being adapted to substantially seal against the inside surface of the piston side wall, each lobe further being adapted, when the shaft is rotated, to progressively cover or uncover the adjacent flexion and extension ports, to thereby separately and simultaneously control flow area through the flexion and extension ports.

In use, one rod of the piston is connected to one segment of the AKP and the far end of the cylinder is connected to the other segment. For purposes of this description, It Is assumed that the upper push rod of the damper piston is pivotally connected to the upper leg segment of the AKP and the lower end of the cylinder is pivotally connected to the lower leg segment. Therefore, in flexion the damper will contract and thus the piston will be driven downwardly in the cylinder by body load. In extension, the damper lengthens and the piston is pulled upwardly by body action.

In the operation of the damper:

- If tho valve is positioned to enable flexion and if the piston is forced downwardly, thereby pressurizing fluid in the lower end of the cylinder chamber, fluid will flow upwardly through the lower check valve and extension ports, if open, into the piston chamber and will leave the piston chamber through the upper flexion ports - fluid will not leave the piston chamber through the extension ports (if uncovered) because there is no significant fluid pressure differential between the lower end of the cylinder chamber and the piston chamber;
- If the valve is positioned to enable extension and if the piston is pulled upwardly, thereby pressurizing fluid in the upper end of the cylinder chamber, fluid will flow downwardly through the upper check valve and flexion ports, if open, into the piston chamber and will leave the piston chamber through the lower extension ports - again fluid will not leave the piston chamber through the flexion ports because there is no significant fluid pressure differential between the upper end of the cylinder chamber and the piston chamber.

It will be noted that the damper design is characterized by the following attributes:

- The valve can be adjusted to vary port areas and thus fluid flow rates to thereby vary resistance to knee joint rotation in either flexion or extension at the same time, thereby enabling variation-of damping in both directions at the same time;
- Because the ports are provided in diametrically opposed pairs, the valve does not get pressed against one side of the piston wall under heavy load and therefore does not seize up or become difficult to move - thus a small motor and shaft can be used to control the damper, which contributes to the compactness and lightness of the unit;
- Because the damper is hydraulic, it is not significantly affected by wear and remains substantially consistent in its damping performance, thereby enabling the user to become accustomed to its "action" and to gain confidence in its performance. One could argue that the temperature of the hydraulic oil could vary and this would affect consistency of performance but this effect is minimized by using aircraft hydraulic fluid.

In a broad aspect, the damper design therefore involves providing:

- a pair of closed chambers (for example the two ends of the cylinder chamber;
- means (for example the piston and cylinder) connected to the leg segments and forming two passageways (for example each formed by a check valve assembly, the piston chamber and a pair of the ports), for moving or pumping fluid from one end chamber to the other through one of the passageways when the leg segments are moving together and through the other of the passageways when the leg segments are moving apart; and
- means (for example the valve and port assembly) for regulating the flow of fluid through each passageway.

In another aspect of the invention, advantage is taken of the repetitive nature of leg actions. If, for example, one is walking along a level surface, there are patterns of knee angle and lower leg strain measurements which do not change significantly from step to step. By monitoring the two sets of signals and timing, the computer software can determine the stage or state of AKP motion and can initiate appropriate changes in flexion and extension capability. If there is deviation from the regular pattern, such as stubbing the AKP toe In the course of swing phase, the software can detect this change and initiate corrective action.

Thus the system incorporates a method for controlling the knee joint of an AKP, which can be stated in the case of level walking as follows:

- storing, in a computer memory, threshold values of lower leg strain and knee angle, which values are indicative of the knee bending in stance phase, of anterior positioning of the center of gravity of body weight relative to the ankle or foot, and of swing phase, all In the course of a step along a level surface;
- continuously sensing lower leg strain and knee angle during use of the AKP and producing electronic signals corresponding thereto;
- comparing the signals against the stored threshold values and, when the signals substantially correlate with threshold values, actuating means for altering the rate of rotation of the knee joint in at least one

of flexion and extension to enable the knee joint to flex at about the beginning of stance phase, to lock the knee joint against flexion while enabling extension in the middle portion of stance phase, and to free the knee joint as it approaches the swing phase, thereby substantially simulating natural knee action; and

- repeating the foregoing repetitiously.

By combining the sensing means, the damper having means which can simultaneously and separately control flexion and extension and the software based on the profiles of repetitive motion, a knee joint system has been evolved which is characterized by closely controlled, predictable responses. This results in the user gaining confidence in the system which then manifests itself in the form of a longer and more rhythmic gait. The software can react similarly whether the gait is fast or slow. And the software can be "fine tuned" to the particular user to gain further compatibility or altered to modify the operation of the AKP. In addition, the system is adaptable to controlling the knee joint in the course of actions other than level walking, such as stair descent and sitting.

From the foregoing, it will be understood that the invention utilizes programmed computer means for receiving the emitted signals from the sensing means, continuously establishing from said signals the state of the AKP in the course of a movement and activating the actuating means to vary damping to substantially simulate natural knee action. More particularly, the programmed computer means is adapted to compare the emitted signals against stored threshold values indicative of transition points between states of a repetitive movement of the AKP and, when the signals substantially correlate with threshold values, to alter the rate of rotation of the knee joint in one or both of flex-on and extension. Preferably, the stored threshold values are selected from the group consisting of the absolute and derivative values of knee angle and the position of the center of gravity of the user's body relative to the AKP foot, the duration from the last transition point and the possible future states in the course of the movement.

The invention described can be thought of as a machine which reacts to the amputee's movements, thus improving gait. Confidence in the machine is necessary for the amputee to take full advantage of the machine's capabilities. This confidence is developed by ensuring that the machine reactions are reproducible, step after step.

In order to obtain consistent and reproducible reactions, the invention takes advantage of the reproducible mechanics of the prosthesis during normal walking. As previously stated, during each step the knee goes through a pattern of movement which is basically the same, step after step. Also reproducible from step to step are the strains on the frame of the AKP, developed by the weight of the amputee, and the angle changes of the knee joint.

The repetitive nature of the signals is an important aspect of the success of the invention. This allows the prosthesis to have consistent man/machine interactions. The prosthesis is a tool used by the amputee to perform different tasks. If the performance of this tool is predictable and reproducible, then user confidence is gained.

With the reactions occurring at the same time and in the same manner for each step, the amputee develops trust in the machine and is able to walk with a continuous fluid motion.

In summary, the invention works on the principle that each step can be divided into segments or states and that a machine reaction can be developed for each segment, thus improving gait. The division of the step is carried out by first obtaining information from the prosthesis, conditioning this information with electronics and analyzing it with software, and then implementing machine reaction by separately and simultaneously varying resistance to flexion and extension rotation of the knee joint.

DESCRIPTION OF THE DRAWINGS

Figure 1 is a block diagram showing the flow of information in the system;

Figure 2 is a perspective simplified view of the Hall effect sensor used for providing signals indicative of knee angle;

Figure 3 is a plot of knee angle sensor output versus knee joint rotation;

Figure 4 is a perspective view of the prosthesis in exploded form;

Figure 4A is a perspective view of the prosthesis in assembled form;

Figure 5 is a plot of strain sensor output versus strain or load on the prosthesis;

Figure 6 is a diagram showing the states in level walking, with the appropriate state conditions shown;

Figure 6a is a diagram showing the states in level walking and correlating them with leg action, piston position and valve position;

Figure 7 is a plot showing the relationship between knee angle and strain (ankle bending moment or load) signals, related to the states, for level walking;

Figure 7a is a diagram showing the states in stair descent and correlating them with leg action, piston position and valve position;

Figure 8 is a diagram showing the states in sitting down, with the appropriate state conditions shown;

Figure 8a is a diagram showing the states in sitting down and correlating them with leg action, piston position and valve position;

Figure 9 is a plot showing the relationship between knee angle and strain signals, related to the states, for sitting down;

Figure 10 is a diagram showing the states in stair descent, with the appropriate state conditions shown;

Figure 11 is a plot showing the relationship between knee angle and strain signals, related to the states, for stair descent;

Figure 12 is a comprehensive diagram showing the states and conditions for the various modes of action;

Figure 12a is a comprehensive diagram corresponding with Figure 12 and showing the various body actions;

Figures 13 and 14 are simplified sectional side views showing the piston and cylinder in flexion and extension modes;

Figure 15 is a simplified end view of the internals of the piston;

Figures 16 - 24 are views similar to Figure 15, showing the valve in various positions;

Figure 25 is a side sectional view of the cylinder and piston;

Figure 26 is an overall circuit diagram of the system;

Figure 27 is a diagram of the communication circuit;

Figure 28 is a diagram of the microprocessor chip;

Figure 29 is a diagram of the voltage references and regulator for the analog to digital convertor located on the microprocessor chip;

Figure 30 is a diagram of the conditioning electronics for the Hall effect sensor;

Figure 31 is a diagram of the conditioning electronics for the strain sensor;

Figure 32 is a diagram of the conditioning electronics for low battery detection;

Figure 33 is a flow chart of the software and Figure 34 is an interrupt service routine which is activated every 20 milliseconds; and

Figure 35 is a perspective view showing strain gauge positioning on the base of the frame.

DESCRIPTION OF THE PREFERRED EMBODIMENT

Having reference to Figures 4a and 4b, the prosthesis A comprises a suction socket 1 which is custom fabricated to closely fit the stump of the amputee and to cling to it by suction. An adjusting plate 2 is attached to the base of the socket 1. A knee bracket 3 is secured by screws to the adjusting plate 2. The knee bracket 3 has apertured shaft supports 3a, 3b for receiving, supporting and affixing the main knee joint shaft 9 and the damper shaft 15 respectively. A frame 4, having a bearing 4a at its upper end, is rotatively mounted to the knee bracket 3 by the main shaft 9, which extends through the bearing 4a. The frame 4 is therefore free to rotate or pivot on the fixed main shaft 9. At its lower end, the frame 4 forms a rectangular socket member 4b for receiving a rectangular block 7a which is clamped to the upper end of the foot pylon 7. Screws secure the pylon block 7a to the frame socket member 4b. A foot 8 is secured to the lower end of the pylon 7.

An upper bearing housing 12 is mounted for rotation on the damper shaft 15. The damper shaft 15 is located to the rear of the main knee joint shaft 9, so that the shaft 15 and upper bearing housing 12 follow an arc relative to the shaft 9 when the knee bracket 3 rotates or pivots.

A Hall effect sensor 13, shown in Figure 2, is provided to monitor the change in knee angle or knee joint rotation. The sensor 13 used is available from Sprague Electronics and is designated as model UGN-3503U. This sensor 13 comprises a ring magnet 11, which is fixed to the stationary damper shaft 15 of the knee bracket 3 by a ring magnet keeper 10. The sensor 13 further comprises a Hall effect transducer 13a, which is located in the rotatable upper bearing housing 12 and which is positioned facing the ring magnet 11. As knee joint rotation occurs, the bearing housing 12 moves around the damper shaft 15, causing the transducer 13a to move relative to the ring magnet 11.

The transducer 13a has a voltage output which is dependent on the magnet flux intensity (north or south pole) directly before it. Therefore, as the knee joint rotates, the output of transducer 13a changes. The signal from the linear Hall effect transducer is amplified to produce .5 volt with a knee joint extended fully and 4.5 volts with the knee joint flexed fully. Included in the circuit is a gain adjustment and an offset control. Stated otherwise, the signal of the transducer 13a is lowest when the knee is straight and increases as the knee is bent. Figure 3 shows a typical sensor voltage output with respect to knee angle after amplification.

The forces on the foot 8 are established by measuring the strain of the frame 4. This is done using foil strain gauges 6 available from Micro Measurements Group Inc., Raleigh, North Carolina under designation CEA - 06 - 062 UW-350. Four gauges 6 are used, two at the front and two at the rear of the frame 4, located between the frame apertures 101 and the base of the frame 4, to measure and differentiate between load on the heel and load on the toe of the foot 8. Stated otherwise, the strain measurement provides an indication as to whether the user body center of gravity is in the anterior, centred or posterior position relative to the AKP foot. The four gauges are wired in a wheatstone bridge configuration to produce an electric signal which changes proportionally with strain. The wheatstone bridge configuration is a standard arrangement for determining the resistance change of strain gauges. The output of the bridge is amplified by a differential instrumentation amplifier 126 to produce an output signal of .5 volts when the heel is loaded fully and 4.5 volts when the toe is loaded fully. No load or similar load on the toe and heel produces 2.5 volts. Included in the circuit is gain adjustment and an offset adjustment. Figure 6 shows a typical voltage output of the bridge with respect to foot loading after the signal is amplified. It will be noted that the load signal decreases as the heel is loaded and increases as the toe is loaded.

A servo motor bracket 14 is secured to the base of the bearing housing 12. A servo motor 16 is mounted within the bracket 14. The motor used is available from Airtronics Ltd. under designation 94737.

An upper spring retainer 17 is mounted on the base of the servo motor bracket 14, for a purpose to be described.

A damper B is positioned between the servo motor bracket 14 and the base of the frame 4.

The damper B comprises a hollow cylinder 26, which is externally threaded. A lower spring mount ring 27 is threaded onto the outside surface of the cylinder 26, for a purpose explained below. A lower bearing mount ring 29 is also adjustably threaded onto the outside surface of the cylinder 26, at its lower end. The ring 29 has radially extending threaded bores 100, normal to its central axis, which fit lower bearing pins 5 which are threaded through apertures 101 in the base of the frame 4. Thus the base of the cylinder 26 is pivotally coupled to the base of the frame 4 by threading the pins 5 into the bores 100 of the ring 29. A lock ring 28, threaded onto the external surface of the cylinder 26, is tightened against the ring 29 to lock it in place.

A lower cap 30 fits into the bore 102 of the cylinder 26 at its lower end and closes the bore. The lower cap 30 is held in place by a snap ring 103. The lower cap 30 carries a circumferential O-ring 104, for sealing against the side wall 105 of the cylinder 26. An aperture 106 is formed through the cap 30. An O-ring 107 is mounted in this aperture 106, sealing around the dummy push rod 25 of a piston 24.

At its upper end, the cylinder 26 has an upper cap 21 which fits into the cylinder bore 102 and is held in place by a snap ring 108. The upper cap 21 also carries a circumferential O-ring 109, for sealing against the side wall 105 of the cylinder 26. An aperture 110 is formed through the cap 21. An O-ring 111 is mounted in this aperture 110, for sealing around the push rod 22 of the piston 24.

The hollow cylindrical piston 24 is positioned in the cylinder bore 102. The piston 24 comprises an open-ended drum 112 having upper and lower end caps 113, 114 screwed thereinto. A push rod 22 extends upwardly from the upper end cap 113, through the sealed aperture 110 in the cylinder cap 21, and is secured to the servo motor housing 14. From the foregoing, it will be noted that the bearing housing 12, servo motor housing 14 and push rod 22 form a train of components connected to the damper shaft 15 and bracket plate 3. Thus as the socket 1 pivots about the main shaft 9, this rotational movement is converted into linear movement of the push rod 22 and piston 24.

A tubular spring 18 extends concentrically around the cylinder 26 between the upper spring retainer 17 and lower spring mount ring 27, for assisting the assembly to increase rate of knee extension during the swing phase of gait. This is useful in enabling increased speed of gait.

The piston 24 and cylinder 26 are shown in simplified form in Figures 13 and 14, with the fluid flows identified by arrows in each of flexion and extension.

The cylinder 26 is a closed or sealed unit and it is filled with hydraulic fluid. The piston 24 carries an external circumferential ring seal 115 for sealing against the side wall 105 of the cylinder 26.

The upper cap 113 of the piston 24 has an aperture 116 opening into the piston chamber 117. A spring-loaded one way check valve 118 controls the aperture 116 and allows pressurized hydraulic fluid to move downwardly from the upper end of the cylinder chamber 119 into the piston chamber 117.

The lower cap 114 of the piston 24 has an aperture 120 opening into the piston chamber 117. A spring-loaded one way check valve 121 controls the aperture 120 and allows pressurized fluid to move upwardly from the lower end of the cylinder chamber 119 into the piston chamber 117.

The check valves used are available from the Lee Company, Westbrook, Connecticut, under designation CKFA 2506205A.

A first pair of diametrically opposed flexion ports 122 extend through the piston side wall 123 at a point above the piston circumferential seal 115. A second pair of diametrically opposed extension ports 124 extend through the piston side wall 123 at a point below the circumferential seal 115.

From the foregoing and having reference to Figure 13, when body weight acts downwardly on the push rod 22 and piston 24, with the flexion ports 122 open, hydraulic fluid may flow upwardly from the lower end of the cylinder chamber 119, through the lower check valve 121 into the piston chamber 117, out of the piston chamber through the flexion ports 122 and into the upper end of the cylinder chamber 119. Therefore, as long as the flexion ports 122 are open, the piston 24 may move downwardly, the damper B may contract and flexion of the knee joint may occur. If the flexion ports 122 are only partly open, there is damping or resistance to the knee rotation in flexion. If the flexion ports 122 are closed, the piston 24 is prevented from moving downwardly and the knee joint is locked against flexion.

Similarly, having reference to Figure 14, when the push rod 22 and piston 24 are pulled upwardly, with the extension ports 124 open, pressurized hydraulic fluid may flow downwardly from the upper end of the cylinder chamber 119, through the upper check valve 118 into the piston chamber 117, out of the piston chamber through the extension ports 124 and into the lower end of the cylinder chamber 119. Therefore, as long as the extension ports 124 are open, the piston 24 may move upwardly, the damper B may extend and extension of the knee joint may occur. If the extension ports 124 are only partly open, there is damping or resistance to knee extension. If the ports 124 are closed, the piston 24 is prevented from moving upwardly and the knee joint is substantially locked against extension.

As previously stated, restriction of the fluid flow through the ports reduces the flow of fluid through the hollow piston, thereby controlling the rate of movement of the piston.

The rate of flow of the fluid is controlled by an adjustable rotatable valve 23. This valve 23 is illustrated in Figures 4, 4b and 16 - 24. It comprises a shaft or rod 36 carrying a pair of lobes 125. The rod 36 extends axially and centrally into the piston chamber 117. It further extends upwardly through a bore 126 in the push rod 22 and is drivably connected with the servo motor 16 housed in the bracket 14.

The lobes 125 extend radially from the rod 36, substantially seal against the inside surface of the piston side wall 123 and each is adapted to extend vertically across both the upper flexion port 122 and the lower extension port 124 on one side of the piston 24.

The associated ports 122, 124 on each side of the piston 24 are circumferentially offset, as shown in Figures 16 - 24. Stated otherwise, the lower extension port 124 begins approximately where the upper flexion port 122 ends. The ports 122, 124 are narrow elongate horizontal slits. Typically they might have a length of .25 inches and width of .02 inches.

Therefore, there is a progressive nature to the reduction and subsequent increase in open area of a port as the valve lobe moves across it on a rotational travel. This of course affects the rate of fluid flow through the piston chamber 117 and determines the relative damping or resistance to rotation experienced by the knee joint.

By circumferentially offsetting the associated pair of upper and lower ports, there is a sequential and separate nature to the opening and closing of flexion and extension ports.

Stated otherwise, and as shown in Figures 16 - 24, the flexion and extension ports of an associated pair of ports on one side of the piston:

- can each be separately progressively opened or closed; or
- each can be separately fully opened or closed; or
- one can be fully closed while the other is progressively closed; or
- both can be fully closed,

and all of the foregoing can be accomplished with a single motor and valve, thereby assisting in achieving compactness and low weight.

The rotation of the inner valve 23 is determined by the software controlling a microprocessor 32, which in turn controls the servo motor 16.

Each step or movement of the prosthesis has been divided into segments (states), dependent on comparison of the incoming sensor signals and preset threshold values. Held in the memory of the microprocessor is a position signal for the inner valve 23. With each change from state to state the inner valve 23 position is altered, thus achieving a different knee joint control. For example, referring to Figure 6A, state No. 1, the initial portion of stance phase, the inner valve 23 is set to allow fluid to escape from the flexion ports 122 and consequently the knee joint can bend as the amputee applies weight. The programmed computer monitors the increasing knee angle and when it reaches the stored threshold value that indicates that the knee has bent to the predetermined angle initiating state No. 2, then the position of the inner valve 23 is altered to completely restrict fluid flow from the flexion ports 122 and allow flow from the extension ports 124. This stops further knee joint bending and allows extension.

The above example illustrates that the assemply can have different control parameters depending on the direction of knee joint rotation (i.e. locked in flexion and allow extension). The fluid passes through separate ports for each of the two directions of knee movement. Therefore, if the flexion and extension ports are restricted independently of each other, the control of the rate of piston movement can be different for each direction.

The Figures 16 - 24 show discrete positions for the inner valve 23. In fact the positioning of the inner valve can be set at any position from 0 to 100 degrees, thus obtaining virtually an infinite range of knee joint damping. This is desirable for "tuning" the leg in activities such as stair descending, where the rate of descent must appeal to the amputee.

The microprocessor 32 used is available from Motorola Semiconductors Ltd. under designation XC 68 HC 811 E2 FN. This is an 8 bit processor having 2K of memory, 8 analog to digital convertors, and 8 digital inputs. The chip is about 1" x 1" and there is no need for any other peripheral chips, thereby allowing it to fit into a small package within the prosthesis A.

The knee angle and load sensor signals are amplified and then fed directly into the microprocessor 32. The amplifiers 126, 127 used for knee angle and load signal conditioning are available from Texas Instruments under designations TLC 272 and TLC 274 respectively.

As shown, the amplifiers 126, 127 and microprocessor 32 are mounted on a circuit board 20 and are enclosed together with a battery 34 (Motorola SNN 4038A) and battery holder 33 in a shell 19 which is secured to the frame 4.

SOFTWARE

The software is set forth in the flow chart and attached Appendix.

Due to the similarities of the sensor information during the course of each step from one step to another (repetitiveness) it is possible to determine the amplitude of each of the two signals at transition points during each step. These transition points are important times when the damping of the knee joint should be altered to allow the amputee to walk. The transition points are detected by the processor 32 by comparing the predetermined "threshold" values, stored in memory, with the real signals from the prosthesis A and cycling through the transition points as they occur. As long as the amputee continues to produce signals as expected, the processor can keep track of the cycle.

With this type of software in operation the hydraulic damper B can be adjusted as each transition point occurs, to a new position which was predetermined during fitting.

This system can therefore determine the position of the prosthesis A during the course of each step and apply an appropriate damping coefficient to the knee joint. Furthermore it is possible to detect whether the amputee is walking on level ground, down stairs, sitting down or has encountered a dangerous situation such as the toe of the prosthesis hitting the ground during swing phase (toe stubbing).

Level Ground

Figure 6A illustrates the point. Each of the numbered circles are referred to as states. The processor always begins in state #1 where the step begins. As the amputee applies weight to the prosthesis A the knee joint begins to bend. This increases the knee angle signal which is continuously being compared to a preset threshold value and as it equals or exceeds the threshold value the processor cycles to state #2. The hydraulic damper setting is altered at the transition point to predetermined settings to allow knee flexion while in state #1 and to lock knee flexion while in state #2.

During state #1 the damper's function is to damp knee flexion and simultaneously allow knee extension and during state #2 to lock knee flexion and simultaneously allow but damp knee extension. Note that the flexion damping has gone from a damped setting to a locked setting independent of the damped knee extension setting. This design allows the amputee to straighten the knee during state #2 even though the knee flexion is still locked.

The damped setting is required to control the rate of knee extension as the amputee proceeds. If a free extension setting was chosen the knee would "snap" straight giving the amputee a noticeably abnormal gait.

The initial knee flexion after heel contact and the straightening of the knee is found in normal gait patterns and is referred to as "knee bounce".

The exact mechanics as to how the hydraulic damper functions is shown in Figure 6A beside each numbered circle.

Figures 6 and 7 show the rules used for the comparison and the actual values of the output of the sensors expected for one step. Following through the step it can be seen that the transition from state #1 to state #2 occurs as the knee angle signal in Figure 7 increases.

The graph shows that knee flexion stops shortly after the transition to state #2. The time delay is the time required for the damper to change.

As the amputee proceeds through the step the next important event is swing phase (time while the foot 8 is off the ground). Indication of the oncoming swing phase can be detected by continuously monitoring the load signal and comparing it to a predetermined value.

As the centre of gravity of the amputee pass over the foot, weight is applied to the toe. The increase in the load signal causes the processor to switch to state #3 as soon as the load signal is equal to or exceeds the predetermined threshold value. The damper is commanded to unlock the knee joint, thus allowing the amputee to initiate swing phase when ready.

The entire swing phase is tracked by the processor. The transition to state #4 occurs when the knee signal increases past a preset threshold value as the knee joint flexes during the initial portion of swing phase.

After state #4 the strain or load signal is ignored and the processor monitors the first derivative of knee angle. The derivative is an indication of the speed and direction of the knee rotation. As the knee joint reaches the maximum flexion during swing the derivative becomes zero and detection of this produces a switch to state #5. Note that the same command for the damper is maintained throughout states #3-4-5, that is, free flexion and free extension which allows swing phase to be completed.

Completion of the swing phase is detected when the knee angle signal decreases past a preset threshold value to indicate that the knee joint has extended back to the straight position. The processor switches to state #1 and the entire process is repeated as long as the amputee continues to walk on level ground.

Emergency Swing Phase Recovery (Stubbing the Toe)

The normal repetitive pattern of knee angle and strain information causes the processor to cycle through state #'s 1-2-3-4-5-1 (see Figures 6 & 7). When the toe of the prosthesis has contacted an obstacle during the swing phase the pattern is different. The pattern is now 1-2-3-4-5-6-1. After state #5 the processor monitors the knee angle derivative information and switches to state #6 if the first derivative has become positive, indicating that the knee is no longer extending but is now flexing (i.e. the obstacle has interrupted the normal velocity of the knee extension). During state #6 the damper is instructed to lock the flexion of the knee joint.

Additional state changes exist for the level walking diagram. Circumduction is the completion of the swing phase without flexing the knee joint. This is done by swinging the limb sideways in an arc to clear the ground instead of flexing the knee. Without the flexion of the knee during the swing phase the processor would switch from state #'s 1-2-3 and stop. This problem is alleviated by measuring the time that the processor is in state #3 and if the knee has not been flexed in a predetermined amount of time the processor switches back to state #1 regardless of any inputs.

Sit Down Mode

During the daily events there are times when the amputee is sitting for an extended period of time. The knee joint of the prosthesis should be in an unlocked position for this time in order for the amputee to position the leg in any desired position. For instance he may wish to have it flexed to place the foot under a chair, or in a right angle position to sit upright, or in a partially flexed position for sitting in a car. The positioning is done by manipulating the prosthesis usually with the hands or the contralateral (other) foot.

Sitting is accomplished by training the amputee to perform a certain move to instruct the processor of the attempt to sit down. Figures 8A and 8 show the cycle of states for sitting down. Figure 9 shows the change in signals for a typical sit down motion. Initially the processor will be residing in state #1. The amputee leans backward which increases the load on the heel of the prosthesis and begins to flex the knee joint. The processor switches from state #1 to state #2 as the knee signal passes a preset threshold value (see state change on Figure 9).

The load on the heel decreases the load signal past a preset threshold value and the processor switches to state #7. As soon as the processor switches to state #7, a timer starts and measures the time which the load is present on the heel. After 1/3 of a second the processor switches to state #8 which commands the damper to allow knee joint flexion. The amputee bears weight on the prosthesis and descends to the chair at a controlled rate. Measurement of time is again made and the processor switches to state #9 after 3/4 seconds. This commands the damper to be free in both flexion and extension of the knee joint, allowing the amputee to manipulate the leg to be comfortable in the seated position. The processor will remain in state #9 until the knee joint is extended to the straight position thus decreasing the knee angle signal past a threshold value at which the processor switches to state #1.

Stair Descending

The usual method for an amputee to descend stairs is to use only his good leg to lower his body weight down each stair until his prosthesis contacts the next stair. He then repeats the motion again using the good leg. The prosthesis is not used at all and the descent is "one stair at a time".

The second method is for the more agile amputee and consists of the normal "step over step" approach but doing so with the knee having uncontrolled descent as his weight flexes the knee (jack knifing).

The present invention incorporates a method of first detecting the fact that the amputee is about to descend a step and then offering a controlled rate of descent.

In order to initiate the descending of stairs, the processor must receive the appropriate signals from the user. This is done by placing the heel of the prosthesis on the edge of the stair and applying weight. Similar to level walking the first state change is from state #1 to state #2 as the knee begins to flex (see above). At this point the load signal decreases (heel loading) and the processor switches to state #7 and then to state #10 as the load reaches a preset threshold value (see Figures 10 & 11).

Note that the amount of weight placed on the heel by the user determines whether the processor stops at state #7 (detects "sit-down") or continues to state #10 (detects "stairs"). The user is trained to apply the appropriate weight to instruct the processor correctly.

A timer is started when the processor switches to state #10. As long as the user maintains the load for 1/2 second the processor will then switch to state #11. During state #11 the damper is commanded to damp the flexion of the knee joint and allow extension. This damping is similar to the hydraulic control unit on a door. The rate at which the door can swing is controlled by the hydraulic fluid within the cylinder. For the knee this damping is preset dependent on the wishes of the user. Some like to descend stairs at a slow rate while others prefer a fast descent.

At completion of each stair the user descends the next step on his contralateral (other) limb. During this time the processor is waiting for the knee joint to extend during the swing phase. The extension reduces the knee signal past a preset threshold value and the processor switches to state #12. The damper is commanded to lock flexion and allow extension. The user again places the heel on the next stair and repeats the sequence 7-10-11-12 for each step. Note that the processor does not return to state #1 after each step. This is due to the lack of a complete extension of the leg prior to the next step.

Once the flight of stairs has been completed, the knee joint is extended to the straight position and the processor switches to state #1 as the knee angle is reduced to a preset threshold value. The choice between stairs, sit down or level walking is now available.

Figure 12 shows all of the states grouped together. At the beginning of each step the software detects whether the amputee is proceeding on level ground (state #'s 1-2-3-4-5-1), has stubbed the toe during a step on level ground (1-2-3-4-5-6-1), is sitting down (1-2-7-8-9-1) or is descending stairs (1-2-7-10-11-12).

The amputee need not push any buttons or turn any levers to instruct the processor to change functions for different terrains. Detection is automatically done in real time dependent on the movements of the amputee.

Additional features of the state diagram include a battery life saver. If the amputee stops for more than 3 seconds in states 1, 2 or 9 the processor stops powering the control motor and goes to a shutdown state.

A low battery warning beeper signals the user that battery replacement is required. In the event that the battery is completely depleted the damper is commanded to damp flexion and free extension prior to complete loss of power. This allows the amputee to still bear weight on the leg without excessive knee flexion until a charged battery is placed in leg. As the flexion is damped the swing phase must be accomplished by circumduction during this time.

APPENDIX

```

;************ text formats ***********
;ALLCAPS: main entry point or CONSTANTs
;lower:    local labels
;Mixed     Ram variables
;name. subr
;NAME| main commands dispatched
```

```
                ; generally a blank line follows changes in flow, or logical thought,
                ; or before entry points.
                ;*****

                ;External addresses in current knee controller 5/88
                ;Using EVM board defaults to simplify debugging
                ;MC8 has RAM 0-7fff, ROM 8000-ffff, DACs fff0-ffff
                ;          RAM     $0000 - $00FF    256 bytes or 512 in E9 chip
                ;          REGS    $1000 - $10FF    $8000 in MC8 IF needed
                ;          EEPROM  $B800 - $B7FF    512 bytes
                ;          EPROM   $F000 - $FFFF    4k 2732 external
                ;          EPROM   $D000 - $FFFF    12k in 68HC711E9
                ;          EEPROM  $F800 - $FFFF    2k in -A2/E2 cpus
                ;          DACS    $FFF0 - $FFFF    only 8 active, addr wraps.
                ;*****

     0000       RS        equ $0000        ;Ram starts here
     00FF       TOS       equ RS+$FF       ;Top Of Stack for -A1 (256 bytes)
     1000       REG       equ $1000        ;Registers sit here

                ;9/02/90 to use EVM for development
                ;can't load EVM EEPROM @ $B600 directly with S records so we're stuck here
     0001       CHIPA2    equ 1            ;if using -A2/E2 for code

     [01]                 ifndef CHIPA2    ;if not -A2/E2
                EE        equ $B600        ;EEPROM starts here in -A1/E9 version
                EES       equ $F800        ;Rules stored here for 1st time up.
                ;Bootup copies to EE if difference in 1st 2 bytes detected (date).
                ROM       equ $F900        ; 2732 starts @ $0F000, 2764 @ $0E000 -A2 @ F800
     [01]                 else             ;in a 2k EEPROM device we have...

     F800       EE        equ $F800        ;EEPROM starts here in -A2 version in Single Chip mode

     F900       ROM       equ $F900        ;start code here in single chip mode

                ;EE       equ $E800        ;EEPROM set here in -A2 version in Expanded mode.
                ;ROM      equ $E900        ;start code here in Expanded mode

     F800       EES       equ EE           ;both in same place in -A2's
     [00]                 endif
                ;*****************
                ;***** cpu def'ns. Uses REG value defined above or defaults to $1000 IFNDEF
                ;         INCLUDE d:\a6811\rule\asm\6811reg.asm
 0000                     INCLUDE \A6811\6811REG.ASM
                          .NLIST 0
                ;01/28/91
     [01]                 ifndef REG       ;default if not defined elsewhere
                REG       equ $1000
     [00]                 endif
                ;**** 6811 CPU I/O PORTS
     1000       PORTA     equ REG+$00      ;Port A & Timer Funcs
     1002       PIOC      equ REG+$02      ;port dir/stbs control
     1003       PORTC     equ REG+$03      ;Port C I/O
     1004       PORTB     equ REG+$04      ;Port B Out
     1005       PORTCL    equ REG+$05      ;Port C Latched in
      1007      DDRC      equ REG+$07      ;Data DiR port C
      1008      PORTD     equ REG+$08      ;Port D
```

```
1009     DDRD    equ REG+$09    ;Data DiR port D
100A     PORTE   equ REG+$0A    ;Port E for digital in on A/D

         ;**** TIMER / COMPARE REGS
100B     CFORC   equ REG+$0B    ;Compare force reg
100C     OC1M    equ REG+$0C    ;OC1 action Mask
100D     OC1D    equ REG+$0D    ;OC1 Data
100E     TCNT    equ REG+$0E    ;Timer Counter reg
1010     TIC1    equ REG+$10    ;Input Capture 1
1012     TIC2    equ REG+$12    ;IC2
1014     TIC3    equ REG+$14    ;IC3
1016     TOC1    equ REG+$16    ;Output Compare 1
1018     TOC2    equ REG+$18    ;OC2
101A     TOC3    equ REG+$1A    ;OC3
101C     TOC4    equ REG+$1C    ;OC4
101E     TOC5    equ REG+$1E    ;OC5
1020     TCTL1   equ REG+$20    ;Timer Control Reg 1
1021     TCTL2   equ REG+$21    ;TC reg 2
1022     TMSK1   equ REG+$22    ;Timer MaSK interrupt reg 1
1023     TFLG1   equ REG+$23    ;Timer FLaG int reg 1
1024     TMSK2   equ REG+$24    ;Timer MaSK int reg 2
1025     TFLG2   equ REG+$25    ;Timer FLaG int reg 2
1026     PACTL   equ REG+$26    ;Pulse Acc ConTroL reg
1027     PACNT   equ REG+$27    ;PA CouNT

         ;**** SPI PORT
1028     SPCR    equ REG+$28    ;SPI control reg
1029     SPSR    equ REG+$29    ;SPI status
102A     SPDR    equ REG+$2A    ;SPI data

         ;**** SERIAL COMMUNICATION INTERFACE REGS
102B     BAUD    equ REG+$2B    ;baud rate register
102C     SCCR1   equ REG+$2C    ;SCI control register 1
102D     SCCR2   equ REG+$2D    ;SCI register 2
102E     SCSR    equ REG+$2E    ;SCI status register
102F     SCDR    equ REG+$2F    ;serial communications data register

         ;**** A/D REGS ****
1030     ADCTL   equ REG+$30    ;A to D control register
1031     ADR1    equ REG+$31    ;A/D results
1032     ADR2    equ REG+$32
1033     ADR3    equ REG+$33
1034     ADR4    equ REG+$34

         ;**** CPU CONTROL REGISTERS
1035     BPROT   equ REG+$35    ;Block protect in -E8 flavour
1039     OPTION  equ REG+$39    ;system configuration options
103A     COPRST  equ REG+$3A    ;COP arm/reset
103B     PPROG   equ REG+$3B    ;EEPROM Control
103C     HPRIO   equ REG+$3C    ;highest priority I-bit interrupt and misc.
103D     INIT    equ REG+$3D    ;Ram & I/O mapping
103F     CONFIG  equ REG+$3F    ;Cop/Rom/EEprom enables & EE adr in -A2

         ;*** offsets for indirects
0000     _PORTA  equ $00  ;Port A & Timer Funcs
0002     _PIOC   equ $02  ;port dir/stbs control
0003     _PORTC  equ $03  ;Port C I/O
```

```
0004    _PORTB  equ $04  ;Port B Out
0005    _PORTCL         equ $05  ;Port C Latched in
0007    _DDRC   equ $07  ;Data DiR port C
0008    _PORTD  equ $08  ;Port D
0009    _DDRD   equ $09  ;Data DiR port D
000A    _PORTE  equ $0A  ;Port E for digital in on A/D

        ;**** TIMER / COMPARE REGS
000B    _CFORC  equ $0B  ;Compare force reg
000C    _OC1M   equ $0C  ;OC1 action Mask
000D    _OC1D   equ $0D  ;OC1 Data
000E    _TCNT   equ $0E  ;Timer Counter reg
0010    _TIC1   equ $10  ;Input Capture 1
0012    _TIC2   equ $12  ;IC2
0014    _TIC3   equ $14  ;IC3
0016    _TOC1   equ $16  ;Output Compare 1
0018    _TOC2   equ $18  ;OC2
001A    _TOC3   equ $1A  ;OC3
001C    _TOC4   equ $1C  ;OC4
001E    _TOC5   equ $1E  ;OC5
0020    _TCTL1  equ $20  ;Timer Control Reg 1
0021    _TCTL2  equ $21  ;TC reg 2
0022    _TMSK1  equ $22  ;Timer MaSK interrupt reg 1
0023    _TFLG1  equ $23  ;Timer FLaG int reg 1
0024    _TMSK2  equ $24  ;Timer MaSK int reg 2
0025    _TFLG2  equ $25  ;Timer FLaG int reg 2
0026    _PACTL  equ $26  ;Pulse Acc ConTroL reg
0027    _PACNT  equ $27  ;PA CouNT

        ;**** SPI PORT
0028    _SPCR   equ $28  ;SPI control reg
0029    _SPSR   equ $29  ;SPI status
002A    _SPDR   equ $2A  ;SPI data

        ;**** SERIAL COMMUNICATION INTERFACE REGS
002B    _BAUD   equ $2B  ;baud rate register
002C    _SCCR1  equ $2C  ;SCI control register 1
002D    _SCCR2  equ $2D  ;SCI register 2
002E    _SCSR   equ $2E  ;SCI status register
002F    _SCDR   equ $2F  ;serial communications data register

        ;**** A/D REGS ****
0030    _ADCTL  equ $30  ;A to D control register
0031    _ADR1   equ $31  ;A/D results
0032    _ADR2   equ $32
0033    _ADR3   equ $33
0034    _ADR4   equ $34

        ;**** CPU CONTROL REGISTERS
0035    _BPROT  equ $35  ;Block protect in -E8 flavour
0039    _OPTION         equ $39  ;system configuration options
003A    _COPRST         equ $3A  ;COP arm/reset
003B    _PPROG  equ $3B  ;EEPROM Control
003C    _HPRIO  equ $3C  ;highest priority I-bit interrupt and misc.
003D    _INIT   equ $3D  ;Ram & I/O mapping
003F    _CONFIG         equ $3F  ;Cop/Rom/EEprom enables & EE adr in -A2
```

```
                  ;****Some standard constants
      0000        NUL     equ $00         ;Null termination for strings
      0001        EOT     equ $01         ;End of Text
      000A        LF      equ $0A         ;LF
      000D        CR      equ $0D         ;CR
      0011        XON     equ $11         ;^Q
      0013        XOFF    equ $13         ;^S
      001A        EOF     equ $1A         ;^Z
      001B        ESC     equ $1B         ;Esc code
      0018        CAN     equ $18         ;^X
      0020        SPC     equ $20         ;space code

                          .LIST
0000                      END             ;of definitions

                  ;*****************

      0004        SDRATE equ $4           ;debug step rate

      [01]                ifdef MHZ7      ;if 7.3728Mhz ...
                  ;**** Servo specific values
                  BRATE   equ $12         ; $12=9600 $??=300 @7.3728Mhz /3/4
                  RTCRAT equ 36864        ;real time clock =20mSec for 7.3728Mhz xtal
                  MDEG    equ 22          ;Approx scale tic to time for 7.3728
      [01]                else
      0030        BRATE   equ $30         ;define default com rate $30=9600B $34=600B @8Mhz
      9C40        RTCRAT equ 40000        ;real time clock 20mSec
      0014        MDEG    equ 20          ;Approx scale tic to time
      [00]                endif

                  ;**** Terminal  I/O Codes
      002A        PROMPT equ '*'          ;prompt char
      003E        GRS     equ '>'
      003C        LTS     equ '<'
      003D        EQS     equ '='
      000D        EOL     equ CR          ;cr on input

                  ;*****  EEROM use in single chip mode -A2
                  ;0000 000F       Date, Bdrate... CHNTBL
                  ;0010 01FF       Rule tables (up to code lower limit), currently 00FF


                  ;***** RAM use
                  ;00 - 0F Copy of EE being adjusted. Must Save to make changes permanent
                  ;10 - 1E Sroadr, Currnbf, Currul, Scnrul, Outadr, Frcrul, Frctim,
                  ;20 -      EMPTY
                  ;3E        Beeper timer (don't move, dup defn in rules for separate assembly).
                  ;40 - 4F Main Flags & ram variables
                  ;50 - 6F Analog FIFO 24 bytes + derivatives (8)
                  ;70 - 7F Timer OCx reloads & phases (yet to implement)
                  ;80 - BF I/O buffers & vars   ;INBUF_STA used by pbee for EESAVE ram routine

                  ;C0 - FF stack space (E0+ mini) !!Each int uses 9 bytes of stack!!
                  ;*****

                  ;***** Rule work space *****
0000                      org RS   ;workspace for tuning/adjust
```

```
        0010          RSIZ     equ 16           ;this many to store in EE at a time
   0000               Wrkrul   ds RSIZ          ;space for copy of whatever we're adjusting

                      ;*****
                               org RS+$10     ;start of ram tables
   0010                        
   0010               Srcadr   ds 2      ;source address of rule # (or data) in ram
        0012          curmbf   equ $     ;fix assembler stupidity (DO NOT MOVE Curmbf. BRULxx fixed ref
                )
   0012               Curmbf   ds 4      ;where active mode bit fields stored (32 MAX rules for now)
        0016          stime    equ $     ;stupid assembler
   0016               Stime    DS 2      ;shutdown timer (battery) DO NOT MOVE Stime. BRULxx fixed ref)

   0018               Curmod   DS 1      ;currently active mode. Currul MUST follow
   0019               Currul   DS 1      ;currently active rule.
   001A               Outadr   DS 2      ;address of motor value for fired rule (gtime)

   001C               Frcrul   ds 1      ;Forced rule. MUST preceed Frctim
   001D               Frctim   ds 1      ;Rule timer, MUST follow Frcrul. (SEARCH) & inits STD X

   001E               Inhrul   ds 1      ;Rule to inhibit for time specified. Inhtim MUST FOLLOW
   001F               Inhtim   ds 1      ;Inhibit a rule timer. Inhrul MUST PRECEED
   0020               Scnrul   DS 1      ;scan rul counter (SEARCH) MUST follow Inhtim

   0021               Oc1time  DS 2      ;OC1 period. Basic sample rate (not yet used)

   0023               Rtime    DS 2      ;Beeper timer register

   0025               Offadr   DS 2      ;address offset for print outs

                      ;Reg     ds 2      ;REG address for fewer bytes of LDX #REG (9 refs) not used yet


                      ;***** Main vars in RAM *****
   003D                        org RS+$3D
        1008          BPORT    equ PORTD        ;beeper port
        0020          BBIT     equ $20  ;bit in PORTD controlling beeper (D5=*SS)
   003D               Btime    ds 2     ;beeper command bytes (chg+cmd) Ctime MUST follow
   003F               Ctime    ds 1     ;count of beeps MUST Follow Btime

   0040               Flag1    ds 1             ;Bit flags & constants. Flag2 - 4 MUST FOLLOW in RAM
        0080          XFRFLG   equ $80          ;Xoff received
        0040          EFLG     equ $40          ;input echo on
                      ;LFLG    equ $20          ;line feeds
                      ;HFLG    equ $10          ;hex input          ;not used since commented out in Hex-b
                in

                      ;NFLG    equ $02          ;negative on input  ;not used. commented out in he
                x-bin
                      ;DBFLG   equ $01          ;debug print outs


   0041               Flag2    ds 1             ;Bit flag & constants. MUST follow Flag1!!
                      ;RRFLG   equ $80          ;Rule in RAM being used, DON'T OVERWRITE unless clear
                !
                                                ;not needed since TIMINT ISR scans up to fired rule
                                                ;before returning.
        0080          HLTFLG   equ $80          ;halt the rules (batt test)
        0040     FRFLG   equ $40        ;just fired a rule, shortens scan so glitches & loops
                                        ;visible and allows FORCER a quick exit when done
                                        ;basically one rule fires per TIMINT.
        0020     MFLG    equ $20        ;lock mode changes (must use FLAGS keyboard command)
        0010     FFLG    equ $10        ;we're Forcing a rule flag (forcer)

        0008     CFLG    equ $08        ;Change of rule #, bkgnd prints new # if SET & clrs bi
              t
        0004     PFLG    equ $04        ;Path found flag (scan)
        0002     SFLG    equ $02        ;Scanning rules (timint) flag, prevents recursive ints

        0001     TFLG    equ $01        ;Tune flag. No state changes permitted if SET


  0042           Flag3   ds 1           ;Bit fields for rules to use. MUST PRECEED Flag4
  0043           Flag4   ds 1           ;Spare flags, cleared at power up MUST FOLLOW Flag3

  0044           Msbb    DS 1      ;Most significant byte of keyboard input values
  0045           Lsbb    DS 1      ;Lsb " " ". MUST FOLLOW Msbb

  0046           Lcnt    DS 1      ;line count, MUST precede Llth
  0047           Llth    DS 1      ;line length for MEMDMP, MUST follow Lcnt
  0048           Acnt    DS 1      ;where print header function starts counting

  0049           Dtemp   DS 2      ;Derive scratch space (addd). @@@ use stack??

  004B           Adjadr  DS 2      ;address of memory being adjusted (ADJ sets, TUNE uses)

  004D           Tmot    DS 1      ;Temporary motor output value during tuning

  0050                   org RS+$50
        0010     DSIZ    equ 16    ;index to start of newest data
  0050           Fifo    ds DSIZ   ;Past 2 blocks A/D data. Lowest=oldest A0, Highest=newest A7.
  0060           Anldat  ds 16     ;Current analog data & derivatives saved here
                                   ;@ could save bytes if derivatives only on 4 channels


                 ;setup in advance of changes to TIMINT to handle 4 trains simultaneously
  0070                   org RS+$70
                 Mtime             ;Motor time. Same as OC2
  0070           Oc2time DS 2      ;OC2 PW current motor timer value (LD_MOTOR & TIMINT)
  0072           Oc3time DS 2      ;OC3 PW
  0074           Oc4time DS 2      ;OC4 PW
  0076           Oc5time DS 2      ;OC5 PW
  0078           OC2ticp DS 2      ;OC2 tick/phase
  007A           OC3ticp DS 2      ;OC3 tick/phase
  007C           OC4ticp DS 2      ;OC4 tick/phase
  007E           OC5ticp DS 2      ;OC5 tick/phase

        0080     RAMEND:           equ $     ;visible marker
                         .page
```

```
                   ;**** I/O Buffer definitions
                   ;@@@ this can be reduced to byte pointers & indexing since ram < 256.
                   ; redesign En-que/de-que to be dynamic sized and placed! NO INBEDED CONSTANTS!

                   ; But first we get the sucker to work.

0080                        org RS+$80        ;I/O Vars start here
      0080         BUFFS    equ $             ;Buffers start here
      0020         OUTSIZ   equ $20           ;Output size, 32 bytes or so

0080               OUT_BFSTA      ds OUTSIZ                    ;C start of buffer address
                   ;OUT_BFSTA     equ BUFFS                    ;C start of buffer address
      009F         OUT_BFEND      equ OUT_BFSTA+OUTSIZ-1                  ;C end of buffer address
      001F         OUT_BFMAX      equ OUT_BFEND-OUT_BFSTA                 ;C almost full circular buffer

      0000         OUT_BFMIN      equ $00                           ;C almost empty CB

      00A0         IOSTR    equ $             ;Vars started here

00A0               Out_hdptr      DS 2    ;Out buffer head pointer
00A2               Out_tlptr DS 2    ;Out tail pointer. Points one behind valid data

00A4               Out_bffil DS 1    ;Out buffer fill (In_bffil MUST follow. STD in SCIINI)

00A5               In_bffil  DS 1    ;In buffer fill (MUST follow Out_bffil)

00A6               In_hdptr  DS 2    ;In buffer head pointer
00A8               In_tlptr  DS 2    ;In buffer tail pointer

      00AA         IOEND    equ $             ;ended here
      0016         INSIZ    equ $20-(IOEND-IOSTR)     ;Input size=block size - vars used size
                   ;ie. we steal bytes from input buffer since almost always empty.

      00AA         IN_BUFSTA      equ $                             ;C start of buffer address
      008F         IN_BFEND       equ IN_BUFSTA+INSIZ-1             ;C end of buffer address
      0011         IN_BFMAX       equ IN_BFEND-IN_BUFSTA-4          ;C almost full Circular Buffer

      0000         IN_BFMIN       equ $00                           ;C almost empty CB

                            .page
                   ;**********************
                   ;**** START OF EPROM ****
                   ;**********************
00AA                      relative   ;back to normal to keep branches happy
                   ;********
                   ;subr to update EE from rom if dates differ
                   ;******
      {01}                ifndef CHIPA2      ;if not 2k EE
                          org $E000          ;here for now

                   ERUDATE:
                          ldd #EE            ;destination address for full update
                          std Srcadr         ;starts here
                          ldd EES            ;new ROM date
                          cpd EE             ;with current Date
                          beq rpbret         ;not different (DATES) so no need to change all

                          ldaa #BRATE        ;$F803   ;FROM ROM 1st time.$30=E/13/1>9600 $34=E/13/16
               >600 baud
                          staa BAUD          ;fix it
                          clr $4000          ;clr EVB sci flip flop to enable com Tx
                          cli                ;allow int's so data can get out

                          bsr RUDATE         ;do 256

                          bsr ree2nd         ;next 256

                          sei                ;stop ints until timer/mode inits done
                          rts

                   ;**** subroutine called to update EE with EPROM rules
                   RUDATE:         ldx #uvmsg          ;inform
                          jsr PMSG           ;user

                          ldx Srcadr         ;current source in ram (EE write adr)
                   ;;     beq rpbret         ;nothing to save, AN ERROR
                          cpx #EE            ;where EE sits
                          blo rpbret         ;not EE, skip it!

                          ldy #EES ;rom table
                   ree2nd: clrb              ;256

                   rpcomp: ldaa 0,y  ;get current ram data
                          cmpa 0,x           ;compare with previous saved EE
                          beq rbdec          ;same so skip pgming this byte

                          pshb               ;save counter
                          pshy               ;save current pointer
                   ;dbg...
                          psha               ;save data
                          jsr HOUTC2         ;dbg

                          pula               ;get back
                          psha               ;save again
                          jsr HOUTS          ;dbg

                          pula               ;get data back
```

```
                ;...dbg
                        ldab #$16           ;byte erase first
                        bsr rpbe            ;do it

                        cmpa #$FF           ;did we ONLY NEED an ERASE?
                        beq ?rffdon         ;yes, skip pgmg

                        ldab #$2 ;program
                        bsr rpbe  ;now

                ?rffdon:
                        puly                ;recover em
                        pulb

                rbdec:  inx                 ;bump pointers ahead
                        iny
                        decb                ;and count down
                        bne rpcomp          ;until zero

                rpbret: rts                 ;done update

                ;**** code compression subr
      [02]      rpbe:   ifdef COPON
                        jsr copset          ;reset cop
      [02]              else
      [01]              endif

                        ldy #2200           ;2500 = 10 msec@ 4cy/dey 3333 = 10msec@3/dex  +3/bne


                ;B set to erase or pgm cmd, Y with delay value, and X pointing to address,
                ;A with value to pgm
                ;rpbee:
                        stab PPROG          ;set EELAT
                        staa 0,x  ;write or erase @ x
                        inc PPROG           ;EEPGM up
                ?wt10:  dey                 ;count down
                        bne ?wt10           ;until done
                        clr PPROG           ;finished
                        rts                 ;return to EE code

                uvmsg:  db "Updating E","E" + $80

                ;**** end of ERUDATE
      [00]              endif
                        .page
```

```
F800                          org EES   ;Rules start here

        F800         RULSTRT          equ EE    ;working EE is here
                     ;            INCLUDE D:\A6811\RULE\ASM\BRUL29.ASM   ;set of rules
                     ;            INCLUDE ASM\BRUL29A.ASM         ;new set of rules
F800                              INCLUDE ASM\BRUL30.ASM          ;Kelly's name
                     ; 14:40 11/12/91
                     ; Base Rules BRUL30.ASM
                     ; Example of RULE30 definition based on 01/15/91 basem1 updated to 4/30/91
                     ; 24 rules possible in this configuration, 3 modes supported, all dynamic
                     ; Revison History
                     ; 1/28/91 added battery rule, HDR's
                     ; 2/4/91 MBIAS, BDRATE & SDRATE added in alterable EE
                     ; 2/19/91 Fixed error in Mode table RULE8 has no cond :. DON'T SCAN IT!
                     ; 2/22/91 Increased size to 24 address & added spare byte to existing rules
                     ; 2/28/91 Added demo digital rules
                     ; 3/12/91 Adjusted rule values to new defaults
                     ; 3/37/91 Add structure for Flags in rules
                     ; 4/04/91 Reverse order of Forced rules & timer to add Inhibit rules
                     ; 4/9-12/91 Work on battery rules & refinements
                     ; 4/30/91 KBJ Remove unused rules
                     ; 5/03&9/91 Rework battery rules
                     ; 5/22/91 Add NCOL for debug width
                     ; 11/12/91 Changed some values for four Tony Semps leg
                     ; 11/14/91 rules for seperate sit and stairs

                                  public DATE,NUMRUL,NUMMOD,BFCNT,MBIAS,MODE00,BDRATE,CHNLST
                                  public SUBADR
                     ;everything else should be computable from tables as offsets

                     ;*** Configuration defines
        {01}                      ifndef RULSTRT  ;if not defined, must be seperate assembly
                     ;RULSTRT      EQU $B600      ;likely address if seperate assembly
                     RULSTRT EQU $F800      ;likely address if seperate load
                     BRATE  EQU $30        ;usual baud rate 9600
                     ;BRATE  EQU $32       ;usual baud rate 2400
                     Btime   equ $3D       ;sits here for now
                     ;EE     equ $B600     ;usual if stand alone
                     EE      equ $F800     ;usual if stand alone
                     EES     equ RULSTRT    ;defaults saved here
                     ;      extern Stime,Cumbf     ;fix errors
                     cumbf  equ $12        ;force a fix for DUMB assembler
                     stime   equ $16       ;force a fix for DUMB assembler

        {00}                       endif
        0000         O       equ RULSTRT-EE  ;difference for absolute references

                     ;REVERSE      equ 1   ;define if channel order reversed
                                                   ;May just use CHNLST to change em.

        0080         FIN     EQU $80 ;Add to final rule # in possible routes list

                     ; We need some definitions to aid later specification of Digital rules
        0080         DIG     EQU $80 ;Bit high in first byte of rule if digital bits present in rul

        0080         B7      EQU $80 ;MSB
        0040         B6      EQU $40
```

```
0020        B5     EQU $20
0010        B4     EQU $10
0008        B3     EQU $08
0004        B2     EQU $04
0002        B1     EQU $02
0001        B0     EQU $01 ;LSB
                   ;Digital rule form is:
                   ;DBBBBBBB      ;D=1, B=Bits that matter = 1. 0 = don't care.
                   ;XVVVVVVV      ;Logic state (Value) for selected bits to meet condition.
                   ; these two bytes present only if digital conditions are used. Dig bit (D) hig
               h
                   ; in first byte of rule.

0000        ANL    EQU $00 ;ANaLog. Dig bit low
0040        CHN    EQU $40 ;CHaiN bit high for additional analog conditions
0020        GTV    EQU $20 ;Greater Than Value
0010        LTV    EQU $10 ;Less Than Value
0000        EQV    EQU $00 ;EQal Value (don't use generally. GT or LT safer since exact
                           ;match not reliable on analog inputs.
0030        NEV    EQU $30 ;Not Equal Value
0008        DVT    EQU $08 ;DeriVaTive (time 20 Msec)
007F        NONE   EQU $7F ;Special case, ANL+CHN+NEV+DVT+CH7 is DON'T CARE state
                           ;!!! No value follows a NONE condition. !!!

[01]                   IFNDEF REVERSE  ;If not reversed

0000        CH0    EQU 0  ;Analog input channel numbers
0001        CH1    EQU 1  ;makes redefinition easier
0002        CH2    EQU 2
0003        CH3    EQU 3
0004        CH4    EQU 4
0005        CH5    EQU 5
0006        CH6    EQU 6
0007        CH7    EQU 7
0000        K0     EQU 0  ;Knee
0001        L0     EQU 1  ;Load
0003        BATT   EQU 3  ;battery

[01]                   ELSE        ;reversed

            CH0    EQU 7  ;Analog input channel numbers
            CH1    EQU 6  ;makes redefinition easier
            CH2    EQU 5
            CH3    EQU 4
            CH4    EQU 3
            CH5    EQU 2
            CH6    EQU 1
            CH7    EQU 0

            K0     equ 7  ;Knee
            L0     equ 6  ;Load
            T0     equ 6  ;Temperature Knee?
            T1     equ 5  ;Temperature Load Cell?
            BATT   equ 4  ;battery voltage
[00]                   endif        end conditional

                       .page
```

```
                            ;****** Start of Rule table in EEPROM
                            ; Rules stay in EEPROM unless modified.
                            ; RULRAM bit field determins if a ram copy of a rule is used since ram
                            ; is very scarce in single chip mode.
                            ; Modified rules are saved to EE when a different rule is selected
                            ; for modification.


                            ;********** START OF DATA *****
F800                                ORG RULSTRT
F800   11 14 91                     DATE:   DB $11,$14,$91  ;in HEX !!

F803   30                   BDRATE: DB BRATE  ;Baud rate saved here to permit changes
F804   30                   MBIAS:  DB $30    ;Motor bias stored here. This is current 11/14/91 value
F805   40                   MSAFE:  DB $40    ;Safe motor value used by BATTERY to lockup cylinder
F806   0F                   BRUL:   DB $0F    ;Battery shutdown rule to fire after timeout
F807   31                   BPVAL:  DB $31    ;Beeper value for battery

                            ;NUMber of RULes in table. This determins how high the search proceedes,
                            ;AND how many bytes are set aside for RULEADR's and RULTIM's.
                            ;space for RULADR's is exact=2*NUMRUL
                            ;space for RULTIM is modulo 8, ie. 1-8=1B,9-16=2B,17-24=3B etc.
                            ;Ditto MODE00, etc.

F808   11                   NUMRUL: DB $11  ;last rule # scanned. 24 rules max in this config.
                                            ;Limit of scan.
F809   17                   MAXRUL: DB 23   ;Maximum legal rule # 0-23
                            ;NUMber of MODes. This makes a dynamic upload without relink possible.
                            ;since all addresses can be computed as offsets from RULSTRT. (but aren't yet)


F80A   03                   NUMMOD: DB $03  ;currently 3 supported. Limited by EE space & rule size.
                                            ;7 max.

F80B   03                   BFCNT:  DB $03  ;# bytes in bitfields since we could have more bit fields then

                                            ;NUMRUL would indicate. Applies to MODEs, RULTIMs, RAMRUL

F80C   02                   NCOL:   db $2   ;# columns to print during debug
                            ;CHaNnel LiST allows dynamic redefinition of input channels without having
                            ;to mess with rules. Allows us to fiddle between systems easily
F80D                        CHNLST:
                            ;       DB $10,$32,$54,$76      ;forward sequence
                            ;       DB $67,$45,$23,$01      ;reverse

                            ;**********
F810                                ORG RULSTRT+$10       ;Keep positon stable if addition bytes added
                                                           ;above (0 free)

                            ;RULes with TIMe constraints. Bit field here saves space & speed
                            ;Binary field with MSB of lowest byte = RULE00, LSB = RULE07, etc.

F810   71                   RULTIM: DB %01110001   ;00-07  ;01,02,03,07 have timer
F811   E0                           DB %11100000   ;08-0f  ;08,09,0A have timer
F812   00                           DB %00000000   ;10-17

                            ;RULes with FLaGs as conditions.
```

```
676  F813  00        RULFLG: DB %00000000   ;00-07  ; have flag words
677  F814  00                DB %00000000   ;08-0f  ;
678  F815  00                DB %00000000   ;10-17
679  F816
680                  ;Bit field of active rules for a given mode. Same DEF'N as RULTIM
681                  ;For 24 rules. 3 bytes required.
682                  MODE00: ;$F0        ;NORMAL OPERATION
683  F816  7F                DB %01111111   ;00-07  ;01,02,03,04,05,06,07 active conditions
684  F817  2C                DB %00101100   ;08-0f  ;0A,0C,0D active conditions
685  F818  00                DB %00000000   ;10-17
686
687                  MODE01: ;$F1
688  F819  00                DB %00000000   ;00-07
689  F81A  00                DB %00000000   ;08-0f
690  F81B  00                DB %00000000   ;10-17
691
692                  MODE02: ;$F2
693  F81C  00                DB %00000000   ;00-07
694  F81D  00                DB %00000000   ;08-0f
695  F81E  00                DB %00000000   ;10-17
696
697                  ;@ or use Flag bits to inhibit/enable rules 0E & 0F.
698
699                  ; and so on for additional modes
700
701                  ;NOTE!! A rule's STATE number is determined by its POSITION in THIS table,
702                  ;NOT its name (eg RUL0A). This makes chosing alternate rules easy, just
703                  ;put the alternates' address in the substituted position in table.
704
705  F81F            RULADR:
706  F81F  F851              DW RULE00-0;00-07
707  F821  F855              DW RULE01-0
708  F823  F85F              DW RULE02-0
709  F825  F866              DW RULE03-0
710  F827  F872              DW RULE04-0
711  F829  F876              DW RULE05-0
712  F82B  F87C              DW RULE06-0
713  F82D  F880              DW RULE07-0
714
715  F82F  F888              DW RULE08-0;08-0F
716  F831  F88D              DW RULE09-0
717  F833  F892              DW RULE0A-0
718  F835  F898              DW RULE0B-0
719  F837  F89B              DW RULE0C-0
720  F839  F89F              DW RULE0D-0
721  F83B  F8A4              DW RULE0E-0
722  F83D  F8A4              DW RULE0F-0
723
724  F83F  F8A4              DW RULE10-0;10-17 ETC.
725  F841  F8A7              DW RULE11-0
726  F843  F8AF              DW RULE12-0
727  F845  F8B7              DW RULE13-0
728  F847  F8BF              DW RULE14-0
729  F849  F8C7              DW RULE15-0
730  F84B  F8CF              DW RULE16-0
731  F84D  F8D7              DW RULE17-0
732
```

```
                    ;List of subroutine addresses. Accessed by output values $F8-$FF.
F84F                SUBADR:
F84F  F8DF                              DW SUB00-0
                    ;      DW SUB01-0      ;etc
                    ;users responsibility to NOT call non-existent subroutines!

                    ;List of data tables for output. Accessed by output values $E0-$EF.
                    ;NOT implimented for Kelly's version
F851                DATAADR:
                    ;      DW DAT00,DAT01  ;etc
                           :none

                    ;*** Start of main part of rule table ****
                    ; Maximum number of rules is 128, named from $00 to $7F
                    ; RULE00 is fired at power up and restart.
                    ; It should not be in active mode list.

                    ;RULE00:
                    ;      DB $00+FIN      ;No previous rule (startup) + Final rule in list
                    ;      DB NONE         ;No conditions
                    ;      DB $F0          ;Invoke mode change to 0, but no output value to mot
            or
                    ;      DB $01          ;Rule to start with in new mode. This byte is unique

                                                ;to mode rules ($F0-F7). It IS NOT TIME!

                    RULE00:            ;BATTERY LOW (special rule does not appear to fire)
F851  80                  DB $00+FIN      ;No previous routes
F852  13                  DB ANL+LTV+BATT ;Analog, Less than, Battery
F853  90            BTVAL: DB $90         ;5.28 volts
F854  F8                  DB $F8          ;Call subroutine

                    RULE01:            ;START
F855  05                  DB $05          ;#5    ;possible routes
F856  06                  DB $06          ;#6
F857  09                  DB $09          ;#9
F858  0C                  DB $0C          ;#C
F859  90                  DB $10+FIN      ;#10 + Final
F85A  10                  DB ANL+LTV+K0   ;Analog, Less than, Knee
F85B  09                  DB $09          ;Threshold
F85C  48                  DB $48          ;Output value
F85D  0F                  DB $0F          ;Force rule F if...
F85E  FA                  DB $FA          ;Time limit of 250 * 20 msec = 5.0 sec

                    RULE02:            ;KNEE FLEXION
F85F  01                  DB $01          ;#1    ;possible routes
F860  8F                  DB $0F+FIN      ;#F + Final
F861  20                  DB ANL+GTV+K0   ;Analog, Greater than, Knee
F862  0A                  DB $0A          ;Threshold
F863  57                  DB $57          ;Output value
F864  0E                  DB $0E          ;Force rule E if...
F865  FA                  DB $FA          ;Time limit of 250 * 20 msec = 5.0 sec

                    RULE03:            ;TOE LOAD (KNEE EXTENSION)
F866  01                  DB $01          ;#1  ;possible routes
F867  02                  DB $02          ;#2
F868  0D                  DB $0D          ;#D
```

```
F869  0E                  DB $0E          ;#E
F86A  8F                  DB $0F+FIN      ;#F + Final
F86B  50                  DB ANL+CHN+LTV+KO      ;Chain, Analog, Less than, Knee
F86C  0F                  DB $0F          ;Threshold
F86D  21                  DB ANL+GTV+LO   ;Analog, Greater than, Load
F86E  9A                  DB $9A          ;Threshold
F86F  32                  DB $32          ;Output
F870  01                  DB $01          ;Force rule 1 if...
F871  2A                  DB $2A          ;Time limit of 42 * 20 msec = 0.84 sec

            RULE04:                 ;FLEXION
F872  83                  DB $03+FIN      ;#3 + Final     ;possible route
F873  20                  DB ANL+GTV+KO   ;Analog, Greater than, Knee
F874  1B                  DB $1B          ;Threshold
F875  32                  DB $32          ;Output

            RULE05:                 ;TERMINAL FLEXION
F876  84                  DB $04+FIN      ;#4 + Final ;possible route
F877  58                  DB ANL+CHN+LTV+DVT+KO   ;Chain, Analog, Less than, Derivative, Knee
F878  FF                  DB -$1          ;Threshold
F879  28                  DB ANL+GTV+DVT+KO       ;Analog, Greater than, Derivative, Knee
F87A  E2                  DB $E2          ;Threshold
F87B  32                  DB $32          ;Output

            RULE06:                 ;STUMBLE
F87C  85                  DB $05+FIN      ;#5 + Final     ;possible route
F87D  28                  DB ANL+GTV+DVT+KO   ;Analog, Greater than, Derivative, Knee
F87E  02                  DB $02          ;Threshold
F87F  53                  DB $53          ;Output

            RULE07:                 ;SMALL HEEL LOAD
F880  02                  DB $02          ;#2  ;possible routes
F881  0C                  DB $0C          ;#C
F882  8E                  DB $0E+FIN      ;#E + Final
F883  11                  DB ANL+LTV+LO   ;Analog Less than, Load
F884  3B                  DB $3B          ;Threshold
F885  57                  DB $57          ;Output
F886  08                  DB $08          ;Force rule 8 if...
F887  32                  DB $32          ;Time limit 50 * 20 msec = 1.0 sec

            RULE08:                 ;SIT DOWN (forced)
F888  80                  DB $00+FIN      ;No previous routes
F889  7F                  DB NONE         ;No conditions
F88A  4E                  DB $4E          ;Output
F88B  09                  DB $09          ;Force rule 9 if...
F88C  64                  DB $64          ;Time limit of 100 * 20 msec = 2.0 sec

            RULE09:                 ;SEATED
F88D  80                  DB $00+FIN      ;No previous routes
F88E  7F                  DB NONE         ;No conditions
F88F  32                  DB $32          ;Output
F890  10                  DB $10          ;Force rule 10 if...
F891  FA                  DB $FA          ;Time limit of 250 * 20 msec = 5.0 sec

            RULE0A:                 ;LARGE HEEL LOAD
F892  87                  DB $07+FIN      ;#7 + Final     ;possible route
F893  11                  DB ANL+LTV+LO   ;Analog, Less than, Load
```

```
F894  2A                  DB $2A        ;Threshold
F895  57                  DB $57        ;Output value
F896  0B                  DB $0B        ;Force rule B if...
F897  32                  DB $32        ;Time limit of 50 * 20 msec = 1.0 sec

                   RULE0B:          ;STAIR DESCENT
F898  80                  DB $00+FIN      ;No previous routes
F899  7F                  DB NONE        ;No conditions
F89A  49                  DB $49        ;Output

                   RULE0C:          ;STAIR SWING
F89B  8B                  DB $0B+FIN     ;#B + Final     ;possible route
F89C  10                  DB ANL+LTV+KO   ;Analog, Less than, Knee
F89D  26                  DB $26        ;Threshold
F89E  57                  DB $57        ;Output

                   RULE0D:          ;FALSE HEEL LOAD
F89F  07                  DB $07        ;#7  ;possible routes
F8A0  8A                  DB $0A+FIN     ;#A + Final
F8A1  21                  DB ANL+GTV+LO ;Analog, Greater than,Load
F8A2  40                  DB $40        ;Threshold
F8A3  00                  DB $00        ;Output

                   RULE0E:          ;SHUT DOWN
                   RULE0F:          ;SHUT DOWN
                   RULE10:          ;SHUT DOWN
F8A4  80                  DB $00+FIN     ;No previous
F8A5  7F                  DB NONE        ;No conditions
F8A6  00                  DB $00        ;No output

F8A7  FFFF FFFF FFFF      RULE11: Dw $FFFF,$FFFF,$FFFF,$FFFF        ;some blank space
  F8AD  FFFF
F8AF  FFFF FFFF FFFF      RULE12: Dw $FFFF,$FFFF,$FFFF,$FFFF        ;some blank space
  F8B5  FFFF
F8B7  FFFF FFFF FFFF      RULE13: Dw $FFFF,$FFFF,$FFFF,$FFFF        ;some blank space
  F8BD  FFFF
F8BF  FFFF FFFF FFFF      RULE14: Dw $FFFF,$FFFF,$FFFF,$FFFF        ;some blank space
  F8C5  FFFF
F8C7  FFFF FFFF FFFF      RULE15: Dw $FFFF,$FFFF,$FFFF,$FFFF        ;some blank space
  F8CD  FFFF
F8CF  FFFF FFFF FFFF      RULE16: Dw $FFFF,$FFFF,$FFFF,$FFFF        ;some blank space
  F8D5  FFFF
F8D7  FFFF FFFF FFFF      RULE17: Dw $FFFF,$FFFF,$FFFF,$FFFF        ;some blank space
  F8DD  FFFF

                   ;additional rules would require that RULNUM be raised, and additional
                   ;bit field bytes for RULTIM and MODExx be added & DW's for RULADR's
                   ;


                   ;Subroutines and D/A tables would go here if used.
F8DF               SUB00:
F8DF  7D 00 17                   tst Stime+1     ;running already?
F8E2  26 0E                      bne nobt        ;yes, don't mess up

F8E4  15 12 40                   bclr <(cumbf),$40     ;take rule 1 out of scan
F8E7  4F                  clra         ;no old beep command
F8E8  F6 F8 07                   ldab BPVAL      ;the value to beep
```

```
F8EB  DD 30                           std Btime    ;new beeper command
F8ED  CC 17 70                        ldd #6000    ;shutdown time out
F8F0  97 16                           staa Stime   ;make it run
F8F2  39          nobt:  rts

                  ;SUB01:
                  ;DAT00:
                  ;DAT01:
F8F3              END_BASE:

F8F3                     END


      1003        DIGIN   equ PORTC                    ;for now
      1004        DIGOUT  equ PORTB

                  ;****************
F900                     org ROM ;Code starts here
                  ;****************
                  ;******
                  ; Main Command Jump Table. Consists of Cmd byte followed by address to jsr
                  ; Searched sequentialy so put most likely commands first in list
                  ; 01/11/91 Redefine elements & routines
                  ;******
      0003        Esiz     equ 3     ;current element size for CMDDISP
                  CMDTBL:            ;^S & ^Q are handled by Rxint/Txint
                  ;        db SPC             ;Eat character
                  ;        dw NULL
F900  41                   db 'A'             ;Adjust values in rule table
F901  FD45                        dw ADJUST
F903  44                   db 'D'             ;Debug. State & A/D values in hex
F904  FF80                        dw DUMP
F906  45                   db 'E'             ;enter values
F907  FD2E                        dw ENTER
                  ;        db 'P'             ;Print something (debug aid)
                  ;        dw PRULS
F909  12                   db 'R'-$40         ;^Restart to power-up values
F90A  F8E2                        dw BEGIN
F90C  54                   db 'T'             ;Tune motor values. Stops rules. Adjusts motor
F90D  FD9F                        dw TUNE
                  ;        db 'R'             ;Recall parameter set #
                  ;        dw RECALL
F90F  53                   db 'S'             ;Save RAM to EE
F910  FCAC                        dw SAVE
F912  4D                   db 'M'             ;Memory dump @ address
F913  FECE                        dw MEMDMP
                  ;        db '?'             ;Help. Display list
                  ;        dw HELP
                  ;        db 'F'             ;Free up mode change
                  ;        dw FREMOD
                  ;        db 'L'             ;Lock to a mode
                  ;        dw LOCMOD
F915  46                   db 'F'             ;Flags toggle
F916  FFB3                        dw FLAGS
                  ;        db 'T'-$40         ;^Test
                  ;        dw TEST1           ;test routines
F918  80                   db $80             ;end of table character MUST be $80
```

```

                    ;****** real time table for real time commands
F919                RTTBL:
F919  3C                    db '<'                  ;dec var
F91A  FEFB                          dw DECVAR
F91C  3E                    db '>'                  ;inc var
F91D  FEF7                          dw INCVAR
F91F  1B                    db ESC                  ;escape loop
F920  FE5E                          dw ESCEXIT
                    ;       db ' '                  ;escape
                    ;       dw ESCEXIT
                    ;       db '.'                  ;escape
                    ;       dw ESCEXIT
F922  3D                    db '='                  ;direct assignment
F923  FEFF                          dw SETVAR
F925  2B                    db '+'                  ;inc address
F926  FD80                          dw INCADR
F928  2D                    db '-'                  ;dec address
F929  FD82                          dw DECADR
F92B  3A                    db ':'                  ;set address
F92C  FD73                          dw SETADR
F92E  80                    db $80                  ;end of table

                    ;*****
                    ; RTDSPH. real time dispatch.
                    ; < > inc/dec value,  + - inc/dec address being modified
                    ; Returns to one ABOVE caller if ESC, SPC, ..  entered.
                    ; Echos variable value for each-step
                    ; falls into CMDDISP
                    ; X is callers address on return to MAIN
                    ;*****
F92F                RTDSPH:
F92F  CE F9 19                      ldx #RTTBL          ;Real Time command TaBLe
                    ;FALL INTO CMDDISP...

                    ;******
                    ;CMDDISP. command dispatch from table of input chars = > addresses
                    ;X points to table head, A has char to check for match.
                    ; If $80 at end of table found, sets Carry, CMD not found.
                    ; X & B altered initially. Commands do whatever...
                    ;******
F932                CMDDISP:
F932  C6 03                         ldab #Esiz          ;get element size for loop use
F934  20 01                         bra cmdlp           ;to middle. Fewer branches inside loop is faster

F936  3A            cmdnxt:  abx                    ;add B to X to get to next table entry

                    ;; may be able to use $80 as flag & bvs to find end
F937                cmdlp:
F937  6D 00                         tst 0,x             ;end of table is $80
F939  2B 0A                         bmi cmderr          ;yes. Cmd not found

F93B  A1 00                         cmpa 0,x            ;byte match?
                    ;       bvs cmderr
F93D  26 F7                         bne cmdnxt          ;get next entry

                    ;;      ldab #1                 ;preload for toggle functions
```

```
F93F   EE 01                        ldx 1,x          ;get address of found command (@X+1) into X
F941   3C                           pshx             ;save on stack

F942   DE 4B                        ldx Adjadr       ;preload for smaller RTDSPH code size

                   ;        jmp 0,x          ;and do it. Called procedure does rts to main loop
F944   39                   rts              ;which is really a jump to table entry on stack!

F945   0D                   cmderr:  sec              ;set carry
F946   39                   rts              ;return with error

                            .page
```

```
                      ;*****
                      ;TIMe INT. OC1 ints here to get work done
                      ;*****
F947                  TIMINT:
F947  CE 10 00                       ldx #REG         ;Base
F94A  10 23 7F                       bclr _TFLG1,x,$7F ;OC1 int flg cleared if high, R/M/Write

                      ;since we should be able to do this in under 20msec, we'll reenable ints & prs
                  y
                      :         cli                ;allow others to int (SCI mostly for debugging)

F94D  1C 00 80                       bset _PORTA,x,$80        ;dbg raise flag OC1 for timing
                      :         bsr fsetx ;dbg raise flag, X ok.

                      ;*** FIFO function shuffles old data in ram to make room for new.
F950  3C                             pshx             ;save #reg
F951  CE 00 50                       ldx #Fifo ;start here
F954  C6 08                          ldab #DSIZ/2     ;loop count

F956  1A EE 08                ?mvdta: ldy DSIZ/2,X     ;get 2 bytes from here &
F959  1A EF 00                       sty 0,X          ;stuff here (FIFO runs toward lower mem)

F95C  08                             inx              ;move pointer up
F95D  08                             inx

F95E  5A                             decb             ;count down
F95F  26 F5                          bne ?mvdta       ;until done

                      :         ldx #REG           ;reload register base for speed/size
F961  38                        pulx               ;recover #REG

                      :         bclr _PORTA,x,$80         ;dbg measurement
                      :         bsr fclrx  ;dbg clear flag, X ok.

                      ;@ we may want ONE pass on 4-7 to minimize skew between channels
F962  1C 30 04                       bset _ADCTL,x,$4 ;change A/D to 4-7 repeating
                      :         ldaa #$14          ;change A/D to 4-7 ONCE
                      :         staa _ADCTL,X      ;now

F965  EC 31                          ldd _ADR1,x      ;get ADR1,2 data. A/D must be done after 20msec...
F967  DD 60                          std Anidat       ;stuff 1,2

F969  EC 33                          ldd _ADR3,x      ;get ADR3,4
F96B  DD 62                          std Anidat+2     ;stuff 3,4

                      ;since we should be able to do this in under 20msec, we'll reenable ints & prs
                  y
                      :         cli                ;allow others to int (SCI mostly for debugging)

F96D  EC 16                          ldd _TOC1,x      ;current time of int
F96F  C3 9C 40                       addd #RTCRAT     ;20msec ;@use Ram value?
F972  ED 16                          std _TOC1,x      ;update compare time

                      ;@ this code should be changed to use the OCx to raise the output, oc1 to clea
                  r
F974  DC 70                          ldd Mtime        ;motor width
F976  27 13                          beq nopul        ;zero
```

```

F978  E3 0E                        addd _TCNT,x      ;add to current time
F97A  C3 00 0A                     addd #10          ;compensate for software delay
F97D  ED 18                        std _TOC2,x       ;update ouput compare reg

F97F  86 C0                        ldaa #$C0         ;* OC2 goes high on compare
F981  A7 20                        staa _TCTL1,x

F983  86 40                        ldaa #$40         ;OC2 force bit
F985  A7 0B                        staa _CFORC,x     ;Force OC2 high

F987  86 80                        ldaa #$80         ;next OC2 will go low
F989  A7 20                        staa _TCTL1,x     ;whenever

F98B                nopul:
F98B  0E                     cli               ;can now allow ints here so PW doesn't jitter

                    ;?wtad0: brclr _ADCTL,x,$80,?wtad0 ;wait for A/D done (just in ca
          se)

F98C  EC 31                        ldd _ADR1,x       ;get ADR1,2
F98E  DD 64                        std Anldat+4      ;stuff 4,5

F990  EC 33                        ldd _ADR3,x       ;get ADR3,4
F992  DD 66                        std Anldat+6      ;stuff 6,7

F994  1D 30 04                     bclr _ADCTL,x,$04 ;restart A/D to CH 0-3 for 18Msec from now
                    ;        ldaa #$30         ;change A/D to 0-3 Repeating
                    ;        staa _ADCTL,X     ;now

                    ;        bset _PORTA,x,$80        ;dbg timing
                    ;        bsr faetx ;dbg set flag, X ok

                    ;*** Beeper check
                    ;@ use X if possible & reload later

F997  18 DE 16                     ldy Stime         ;timing beep?
F99A  27 15                        beq ?batst        ;nope

F99C  18 09                        dey               ;count out
F99E  18 DF 16                     sty Stime         ;update ram
F9A1  26 0E                        bne ?batok        ;not YET timed out

F9A3  14 41 80                     bset Flag2,HLTFLG        ;stop rules
F9A6  B6 F8 06                     ldaa BRUL         ;the rule # to force
F9A9  C6 01                        ldab #1           ;in one tic
F9AB  DD 1C                        std Frcrul        ;stuff here
F9AD  C6 05                        ldab #5           ;new beep
F9AF  D7 3E                        stab Btime+1      ;and shut up beeper with a longer chirp

                    ;        clrb              ;need zero
                    ;        bra cbeep         ;and shut up beeper
                    ;        bra ?batok        ;skip onward

                    ;        ldab MSAFE        ;safety value
                    ;        jsr LD_MOTOR      ;do it now
                    ;        bra ?batok        ;skip onward
```

```

F9B1                    ?batst:
                        ;          ldaa Anldat+3      ;the battery channel
                        ;          cmpa BTVAL         ;the threshold in EE
                        ;          bhs ?batok         ;not too low yet

                        ;          ldd #6000          ;2 minutes
                        ;          std Stime          ;setup time out
                        ;          ldaa BVAL          ;get command
                        ;          staa Btime+1       ;startup beeper

                        ;;         ldd BVAL           ;beep value & repeat
                        ;;         std Btime+1        ;cmd & repeat count

F9B1                    ?batok:
                        ;*********************
                        ;**** Beep: handled here
                        ;@ This can be improved (how?)

F9B1  D6 3E                            ldab Btime+1       ;get command byte
F9B3  27 2E                            beq ebeep          ;nothing happening, return

F9B5  D1 3D                            cmpb Btime         ;compare with previous command
F9B7  26 1E                            bne ton   ;command changed. Start with on time, set counter

                        ;?nchg:                   ;no change. must sill be timing
F9B9  18 DE 23                         ldy Rtime          ;get counter
F9BC  18 09                            dey                ;count down
F9BE  18 DF 23                         sty Rtime          ;update
F9C1  26 20                            bne ebeep          ;continue. time not up yet

                        ;          brclr BPORT-REG,x,BBIT,ton        ;gnd if beep currently off, compute on
                   time
F9C3  1E 08 20 10                      brset BPORT-REG,x,BBIT,ton        ;v+ if beep currently off, compute on
                   time

F9C7                    cbeep:
                        ;          bclr BPORT-REG,x,BBIT      ;clear beeper bit
F9C7  1C 08 20                         bset BPORT-REG,x,BBIT      ;clear beeper bit

F9CA  54                           ?toff:  lsrb               ;shift 'OFF' time to low nybble
F9CB  54                                   lsrb
F9CC  54                                   lsrb
F9CD  54                                   lsrb
F9CE  26 03                                bne offok          ;we do have an off time, else we're done.

F9D0  7F 00 3E                     clrbt:  clr Btime+1        ;clear command. Done for now. (fall through for size)
                                                  ;Btime+1=zero prevents beep timing until cmd changes
F9D3  17                           offok:  tba                ;copy
F9D4  3D                                   mul                ;time squared
                        ;          ldaa #10 ;10 *.020=.2 sec tics
                        ;          mul                ;times low order in B
                        ;          std Rtime          ;update counter
                        ;          bra cbeep          ;clear beep

                        ;& we could do repeat count down here
                        ;&         ldaa Ctime         ;get count
                   ;&        beq ?mul10        ;we don't have a count

                   ;&        deca              ;count out
                   ;&        staa Ctime        ;update
                   ;&        beq clrbt ;timed out, stop update, else

F9D5   20 07                           bra mul10         ;continue with common code

F9D7               ton:
                   ;         bset BPORT-REG,x,BBIT       ;turn on beeper bit
F9D7   1D 08 20                        bclr BPORT-REG,x,BBIT      ;turn on beeper bit
F9DA   D7 3D                           stab Btime        ;update command change byte
F9DC   C4 0F                           andb #$0F         ;mask to low order nybble

F9DE   86 0A                 mul10:    ldaa #10 ;10 * .020 = .2sec tics
F9E0   3D                              mul               ;compute
F9E1   DD 23                           std Rtime         ;update/start counter

                   ebeep:                                ;end of beep functions

                   ;*************************
                   ;**** rule timing. Do we need to check SFLG?
F9E3   DC 1C                           ldd Frcrul        ;get forcing info. rule # & time
F9E5   5D                              tstb              ;timer running?
F9E6   27 08                           beq ?nfrct        ;nope, no forced time active

F9E8   7A 00 1D                        dec Frctim        ;count down timer in RAM
F9EB   26 06                           bne ?nfrct        ;not yet timed-out

F9ED   97 20                           staa Scnrul       ;force this rule (A from LDD)

                   ;         ldaa #'T' ;dbg
                   ;         jsr dbg           ;'T

F9EF   8D 5D                           bsr FORCER        ;make it active NOW

F9F1   20 08                           bra endtim        ;skip to end. only one rule fired per Oc1 int

F9F3   86 83                 ?nfrct: ldaa #(HLTFLG + TFLG + SFLG)  ;flags to check
F9F5   95 41                           bita Flag2        ;in here
F9F7   26 08                           bne norule        ;blocked by tune or scan or HALT (batt low)

F9F9   14 41 02                        bset Flag2,SFLG   ;prevent recursive calls to scan

                   ;         bclr Flag1,DBFLG,?dscn      ;dbg
                   ;         jsr CRLF  ;dbg scan
                   ;         ldaa #CR          ;dbg
                   ;         bsr dbg           ;cr

F9FC   8D 0B                 ?dscn:    bsr SCAN          ;scan rules if not recursive interrupt

F9FE   15 41 02              endtim:   bclr Flag2,SFLG   ;clear the scanning flag, or forcer deadlock

FA01               norule:
                   ;         bsr fclr  ;dbg timing
FA01   CE 10 00                        ldx #REG          ;recover base
FA04   1D 00 80                        bclr _PORTA,x,$80          ;always end INT with OC1 cleared
```

```
FA07  3B                                 rti                 ;all done this interrupt (approx 1.5msec total)

                  ;*****
                  ;timing measurement flag for debugging/refinement
                  ;fclr:
                  ;fset:
                  ;          ldx #REG          ;base
                  ;fclrx:
                  ;fsetx:
                  ;          ldaa _PORTA,X     ;get flag
                  ;          eora #$80         ;flip OC1
                  ;          staa _PORTA,X     ;stuff it back

                  ;;         tst _PORTA,X      ;sign?
                  ;;         bpl fs            ;raise since down

                  ;;         bclr _PORTA,x,$80           ;lower OC1 output
                  ;;         RTS

                  ;;fset:    ldx #REG          ;base
                  ;;fsetx:
                  ;;fs:      bset _PORTA,x,$80           ;raise OC1 output
FA08  39                                 rts
                  ;15 bytes vs 10

                  ;*****
                  ;SCAN. Scans rules & generates outputs
                  ;Modifies all registers
                  ;1/29/90
                  ;*****
                  ;          db "SCN"          ;dbg
FA09              SCAN:
                  ;          rts
FA09  7F 00 20                           clr Scnrul          ;clr counter
FA0C  15 41 40                           bclr Flag2,FRFLG    ;clear 'Fired a Rule' flag

                  ;          ldx #MODE00       ;from eprom during debugging
FA0F  CE 00 12                           ldx #Cumbf          ;from ram table

FA12  A6 00                  ?getbf:     ldaa 0,x  ;get active rules bit field
                  ;          beq ?nxtbf        ;skip if none at all. ;low probability, not enabled

FA14  48                     ?shrbf:     asla                ;bits into carry
FA15  24 0B                              bcc ?nrule          ;this rule not active

FA17  3C                                 pshx                ;this rule active, stack 'em
FA18  36                                 psha
                  ;          bsr fset  ;dbg

FA19  8D 36                              bsr GETRUL          ;and check rule conditions
                  ;          bsr fclr  ;dbg

FA1B  32                                 pula                ;recover 'em
FA1C  38                                 pulx

FA1D  12 41 40 16                        brset Flag2,FRFLG,nxt       ;fired a rule. Only one per int so skip out
                                                             ;next call to scan will clear FRFLG
```

```
FA21  4D                                tsta              ;set CCs

FA22  27 05                  ?nrule:    beq ?nxtbf        ;nothing left in reg so get next byte

FA24  7C 00 20                          inc Scnrul        ;next scan rule #
FA27  20 EB                             bra ?shrbf        ;& check for active status

FA29  08                     ?nxtbf:    inx               ;next bit field

FA2A  D6 20                             ldab Scnrul       ;retrieve it
FA2C  C4 F8                             andb #$F8         ;mask out partial counts (0-7)
FA2E  CB 08                             addb #$8          ;increment to next whole byte of BFs
FA30  D7 20                             stab Scnrul       ;put it back

                     ;          tba             ;dbg
                     ;          jsr HOUT        ;dbg

FA32  F1 F8 08                          cmpb NUMRUL       ;hit limit?
FA35  23 D8                             bls ?getbf        ;nope, go check next BF byte

                     ;donerul:
                     ;          ldaa #'s' ;dbg
                     ;          jsr OUT_OC      ;dbg

FA37  39                        nxt:    rts               ;done this pass.

                     ;messages sit here for bsr limits
FA38  CD                        mdemsg:         db "M" + $80      ;mode error
FA39  D2                        rermsg:  db "R" + $80     ;error mesage here for bsr

                     ;*** Subr to get EE address of rules (not ram)
FA3A  F1 F8 09                  GETEDR:         cmpb MAXRUL   ;limit to present maximum rule
FA3D  23 07                             bls ?mok          ;ok (unsigned, but since 7f max, needn't be)

FA3F  CE FA 39                          ldx #rermsg       ;error
                     ;          bsr mdeout      ;indirect jmp to PMSG
FA42  BD FE 71                          jsr PMSG          ;tell 'em

                     ;          ldab MAXRUL     ;limit it to??
                     ;          ldab #0
FA45  5F                                clrb              ;limit to power up rule since we have an error

FA46  CE F8 1F                  ?mok:   ldx #RULADR       ;table start in EE
FA49  58                                aslb              ;*2 for words
FA4A  3A                                abx               ;index into address table
FA4B  EE 00                             ldx 0,x           ;get address into x
FA4D  39                                rts               ;done

FA4E  14 41 12                  FORCER:         bset Flag2,FFLG + SFLG      ;set Forced & Scanning flags..

                     ;          ldaa #'f'  ;dbg
                     ;          bsr dbg         ;f

                     ;we don't want to stack things up if He goofs and gets an infinite loop of
                     ;forced rules so that we jump around & around & around & .....
                     ;so we fall into...
                     ;          jmp GETRUL      ;force the rule search.
```

```
                                            ;will return to caller after rule fires (we hope)
                                            ;since FRFLG set too, getrul returns quickly too.
FA51
                      ;**subr to get rule from table, check conditions, and fire output & timer
FA51                  GETRUL:
                      :         bsr getscn          ;get the rule's address
                      ;***** subr gets address of rule at Scnrul. Sits here for BSRs limits
                      ;getscn:
FA51  D6 20                           ldab Scnrul          ;scanned rule #
                      ;GETADR:
FA53  8D E5                           bsr GETEDR           ;get EE rule address
FA55  9C 10                           cpx Srcadr           ;compare to address we may have in ram
FA57  26 03                           bne ?raok            ;ok

FA59  CE 00 00                        ldx #Wrkrul          ;NO, here is where THIS rule is
                      ;         sec                 ;and tell them so
                      ;?raok:   rts

FA5C  15 41 04                ?raok:  bclr Flag2,PFLG      ;clr path found flag

                      ;         brclr Flag1,DBFLG,getpath  ;dbg printout
                      ;dbg
                      ;         jsr HOUTC2          ;newline & address
                      ;         ldaa Scnrul         ;get rule #
                      ;         jsr HOUTS           ;print it
                      ;         rts                 ;dbg
                      ;         bra getpath         ;dbg
                      ;         db "GP"             ;dbg

FA5F                  getpath:
FA5F  E6 00                           ldab 0,x  ;get paths byte
FA61  C4 7F                           andb #$7F            ;mask out FIN flag
FA63  27 04                           beq ?pzero           ;path of zero speciall, no previous req'd:. Path found

FA65  D1 19                           cmpb Currul          ;Current rule?
FA67  26 03                           bne ?nmtch           ;nope

FA69  14 41 04                ?pzero: bset Flag2,PFLG      ;found a path

FA6C  6D 00                   ?nmtch: tst 0,x              ;FIN flag?
FA6E  08                              inx                  ;% point to next path, only Z CC affected
FA6F  2B 02                           bmi ?lstph           ;yes, last path

                      ;%        inx                 ;point to next path
FA71  20 EC                           bra getpath          ;loop

FA73  12 41 10 04             ?lstph: brset Flag2,FFLG,?fvpth    ;force valid path if set
FA77  13 41 04 BC                     brclr Flag2,PFLG,nxt       ;had NO PATH, RTS
FA7B                  ?fvpth:
                      ;         ldaa #'p'
                      ;         bsr dbg             ;p

                      ;%        inx                 ;next up
                      ;** Check FLAG Bit field & byte
                      ;         pshx                ;save current pointer
                      ;         ldx #RULFLG         ;FLAGS bit field
```

```
                    ;           jsr BFSCN       ;check flags bit fields for rule # in Scnrul

                    ;           pulx            ;recover pointer
                    ;           bcc ?dtst       ;no flags to check. Test for digital condition

                    ;@@ NEED TO ADD a Force flag test here, but no room for code
                    ;@          brset Flag2,FFLG,?fvflg     ;force valid flags if set
                    ;@ also need to change to same style as digital rules. ACTIVE bits & VALUES
                    ;@ could then use subroutine for both too.

                    ;           ldaa 0,x  ;get the data byte
                    ;           anda Flag3      ;AND in current flags
                    ;           eora Flag3      ;flip sense of HOT bits
                    ;           bne nxt         ;condition not matched, RTS

                    ;?fvflg:    inx             ;skip to next condition
FA7B  E6 00                     ?dtst:    ldab 0,x  ;next condition byte
FA7D  2A 13                               bpl anlrul        ;Analog if MSB=0

FA7F                dignul:
                    ;           ldaa #'d' ;dbg
                    ;           bsr dbg         ;d

FA7F  F4 10 03                            andb DIGIN        ;AND in external data with loaded mask
FA82  E8 01                               eorb 1,x  ;flip sense bits as spec'd
FA84  E4 00                               andb 0,x  ;mask them too for robustness

FA86  08                                  inx               ;point to analog next. skip to sense byte
FA87  08                                  inx               ; and analog conditions
FA88  12 41 10 04                         brset Flag2,FFLG,anlget     ;force valid digital, check on analog

FA8C  C4 7F                               andb #$7F         ;mask out DIG bit
FA8E  26 A7                               bne nxt           ;Condition not matched, get next rul (just an RTS)

FA90  E6 00                     anlget:   ldab 0,x  ;get analog conditions

FA92                anlrul:
                    ;           ldaa #'a'
                    ;           bsr dbg         ;a

FA92  C1 7F                               cmpb #NONE        ;check for don't care
FA94  27 4C                               beq fires ;don't care analog cond, so special fire
                                                    ;no treshold follows this condition!
FA96  12 41 10 3F                         brset Flag2,FFLG,nxtcnd     ;force analog

FA9A  C4 0F                               andb #$0F         ;mask to channel bits only
FA9C  3C                                  pshx              ;save pointer
                    ;*****
                    ; here we put CHNTBL translator
      [01]                      ifdef TRANSLATE ;translator enabled
                                tba             ;save for bit tests
                                andb #$07       ;just A/D ch # bits
                                lsrb            ;into nybbles
                                ldx #CHNTBL     ;translate table
                                abx             ;index into table

                                ldab 0,x  ;new value
```

```
                                  bita #$1  ;odd/even nybble?
                                  beq ?chevn        ;even ;change to bne to swap nybbles around

                                  lsrb              ;shift into low nybble, 0 fill (NOT ASR!!)
                                  lsrb
                                  lsrb
                                  lsrb

                        ?chevn:   bita #DVT         ;$8       ;derivative?
                                  beq ?chok         ;no, use as is

                                  orab #DVT         ;raise this bit, (offset CH address by 8)
                        ?chok:    andb #$0F         ;keep within bounds (kill any high nybble residue)
                        ;30 bytes if Y, 27. if X
           [00]                   endif
                        ;***** end of translator

FA9D  C5 08                               bitb #$8 ;DVT?
FA9F  27 03                               beq ?nodvt         ;nope
                        ;         bra ?nodvt        ;dbg

FAA1  BD FB 92                            jsr DERIVE         ;set derivative for THIS channel into ram table

FAA4  CE 00 60                    ?nodvt: ldx #Anldat        ;analog data table
FAA7  3A                                  abx                ;add in offset to channel
FAA8  E6 00                               ldab 0,x ;get data value

FAAA  38                                  pulx               ;recover pointer

                        ;         brclr 0,x,DVT,?nobug       ;dbg only DVT instructions
                        ;         pshx              ;dbg save pointer
                        ;         ldaa 1,x  ;dbg get threshold
                        ;         xgdx              ;dbg swap to X
                        ;         jsr HOUTC2        ;dbg and print
                        ;         xgdx              ;recover B
                        ;         pulx              ;dbg recover pointer
                        ;?nobug:                    ;dbg

FAAB  E1 01                               cmpb 1,x           ;compare with threshold and branch as required


                        ;@ we might try a data table, indexed by 3 bit offset
                        ;currently branch costs 5 * 4 bytes=20 vs 8 * 2 addresses +index overhead
                        ; or
                        ;?nodvt:
                        ;1        pshb              ;save channels
                        ;2        lda #3            ;offset per branch
                        ;1        mul
                        ;3        addd #7b00        ;base
                        ;2        xgdy              ;to Y
                        ;1        pulb              ;recover channels

                        ;         ldx #Anldat       ;analog data table
                        ;         abx               ;add in offset to channel
                        ;         ldaa 0,x  ;get data value

                        ;         pulx              ;recover pointer
```

```
                         ;          cmpa 1,x            ;compare with threshold and branch as required
                         ;
                         ;3         jmp 0,Y             ;check them
                         ;
                         ; = 13 + 6 for 2 more beq/bne, rts for dvt, may not be faster either.

FAAD  1E 00 30 19                   brset 0,x,$30,?b110      ;this way NEV. MUST test BOTH bits FIRST
                                                        ;ok for DVT too
FAB1  1E 00 18 18                   brset 0,x,$18,?b011      ;this way LTV DVT
FAB5  1E 00 10 08                   brset 0,x,$10,?b010      ;this way LTV = BLO
FAB9  1E 00 28 13                   brset 0,x,$28,?b101      ;this way GTV DVT
FABD  1E 00 20 06                   brset 0,x,$20,?b100      ;this way GTV = BHI
                         ;          brclr 0,x,$30,?b000 ;this way EQV, falls through.
                                                        ;ok for DVT also
                         ;We use hard coded values rather than defined constants
                         ; since code sequence dependencies exist,
                         ;but the following branch chain sorts it all out
FAC1  27 16                 ?b000:  beq nxtcnd          ;condition ok, check for more
FAC3  39                            rts                 ;condition NOT MET, try nextrul

FAC4  25 13                 ?b010:  blo nxtcnd
FAC6  39                            rts

FAC7  22 10                 ?b100:  bhi nxtcnd
FAC9  39                            rts

FACA  26 0D                 ?b110:  bne nxtcnd
FACC  39                            rts

FACD  2D 0A                 ?b011   blt nxtcnd          ;signed for DVT
FACF  39                            rts

FAD0  2E 07                 ?b101   bgt nxtcnd          ;signed for DVT
FAD2                     nomc:
FAD2  39                    nodbg:  rts

                         ;this is here to allow bsr's for debugging
                         ;dbg:       brclr Flag1,DBFLG,nodbg     ;no debug
                         ;          jmp OUT_OC          ;echo chars & return

                         ;sits here for bsr's
FAD3  CE FA 38              mth:    ldx #mdemsg         ;@ERROR ERROR! mode too high for table
FAD6  7E FE 71              mdeout: jmp PMSG            ;so print out, and abort change


                         ;*****
FAD9  1F 00 40 04           nxtcnd: brclr 0,x,CHN,firea         ;chain? = $40 No, fire analog if we got this far

FADD  08                            inx                         ;yes, point to next set (cond 1)
FADE  08                            inx                         ; (anl 2)
FADF  20 AF                         bra anlget                  ;and check it

FAE1  08                    firea:  inx                 ;point to output byte (last cond)

                         ;special case, no conditions
FAE2  08                    fires:  inx                 ; (output)
```

```

                    ;       ldaa #'i'  ;dbg
                    ;       bsr dbg         ;'ignite

                    ;;      ldaa Scnrul     ;we want to be here
FAE3  DC 1F                         ldd Inhtim      ;we want to be here (B=Scnrul follows Inhtim)
                    ;;      cmpb Currul     ;we are currently here
                    ;;      beq ?nfire      ;same as last time so skip firing calls

FAE5  4D                            tsta            ;Inhtim is? (A of LDD Inhtim)
FAE6  27 05                         beq ?kfire      ;no inhibits

FAE8  D1 1E                         cmpb Inhrul     ;does our scan rule match the inhibited rule?(B=Scnrul
              )
                    ;       bne ?kfire      ;nope
FAEA  20 01                         bra ?kfire      ;dbg we don't care

FAEC  39                    ?nfire: rts             ;we didn't REALLY fire the rule. Look at next one
FAED                ?kfire:
                    ;       cmpb #1         ;special rule?
FAED  5D                            tstb            ;special rule? 11/12/91
FAEE  27 08                         beq nosee       ;yes, don't let anyone (except SPI) see it fire

FAF0  D7 19                         stab Currul     ;so tell everyone of new rule number
FAF2  7F 00 1D                      clr Frctim      ;new rule, clear timer.
                    ;       clr Inhtim      ;and inhibits? If inh & other fires, do we stay inh'd?

FAF5  14 41 48                      bset Flag2,CFLG+FRFLG   ;tell bkgd & scan of rule number change
FAF8  15 41 10                      bclr Flag2,FFLG ;and clear forcing flag
                    ;@ there is a logical inconsistency here. Fflg cannot be cleared here,
                    ;@ and then used for testing just before domodel
                    ;@ No mode change must mean no mode change on forced rules too.
                    ;@ an interesting trap. So how do you get into a mode you want to lock to?

                    ;*** write rule # to D/A port via SPI
FAFB  B6 10 29              nosee:  ldaa    SPSR            ;;Knock down any SPI flags
FAFE  B6 10 2A                      ldaa    SPDR            ;;and dump any read data

                    ;       ldab Currul             ;get the rule that fired (stab Currul above)
FB01  58                    aslb                    ;; *2, 128 max (scale up output)
FB02  58                    aslb                    ;; *4, 64 max
FB03  58                    aslb                    ;; *8, 32 max
FB04  F7 10 2A                      stab SPDR               ;; write to SPI port.
                    ;*****
                    ;       tba             ;dbg
                    ;       jsr HOUTS       ;dbg

FB07  DF 1A                         stx Outadr      ;save address of output value for gtimer

FB09  EC 00                         ldd 0,x         ;get output value (A) & byte following (B)
FB0B  81 FC                         cmpa #$FC       ;lower than?
FB0D  25 17                         blo notspl      ;not special

FB0F  27 08                         beq flgset      ;flagset function ($FC)

FB11  81 FE                         cmpa #$FE       ;Beeper?
```

```
1638  FB13   22 36                               bhi digou .        ;Digitall ($FF)
1639
1640  FB15   25 09                               blo flgclr  ;flagclr function ($FD)
1641                         ;BEEPER                          ;equal ($FE)
1642  FB17   4F              ?beep:     clra                  ;clear out old command (@Btime)
1643  FB18   DD 3D                               std Btime          ;startup beep timer with data byte B & no old command
1644  FB1A   20 32                               bra dcont          ;and continue with common code
1645
1646  FB1C   DA 42                      flgset:  orab Flag3         ;"OR" in Flag bits
1647  FB1E   20 02                               bra fcont          ;and continue with common code
1648
1649  FB20   94 42                      flgclr:  anda Flag3         ; "AND" in Flag bits
1650  FB22   D7 42                      fcont:   stab Flag3         ;stuff 'em back
1651  FB24   20 28                               bra dcont          ;and continue with common code
1652
1653  FB26                   notspl:
1654  FB26   81 F0                               cmpa #$F0          ;specials?
1655  FB28   25 37                               blo domot          ;nope, standard motor
1656
1657  FB2A   81 F8                               cmpa #$F8          ;subs? with whats left
1658  FB2C   24 25                               bhs dosubs         ;yes
1659
1660                         ;fall into.domode   ;$F0-$F7
1661                         :          brset Flag2,FFLG,domode     ;forcing a rule?
1662  FB2E   12 41 20 A0                         brset Flag2,MFLG,nomc        ;no mode change allowed
1663
1664  FB32                   domode:
1665                         ;          ldaa #'m'
1666                         ;          bsr dbg               ;m
1667                                                          ;change mode table. called by LOCMOD also (maybe)
1668  FB32   80 F0                               suba #$F0          ;subtract base
1669  FB34   81 F8 0A                            cmpa NUMMOD        ;limit is?
1670  FB37   24 9A                               bhs mth            ;something wrong, over table so don't change yet
1671
1672  FB39   97 18                               sta Curmod         ;update mode prompt
1673                         ;ldd...    ldaa 1,x  ;get 1st rule in new mode
1674  FB3B   D7 20                               stab Scnrul        ;1st rule in new mode will be forced later
1675
1676  FB3D   F6 F8 0B                            ldab BFCNT         ;get bit field count
1677  FB40   3D                                  mul                ;mode # * size/mode in bytes
1678
1679  FB41   C3 F8 16                            addd #MODE00       ;add in base of 1st table in EE
1680  FB44   8F                         xgdx                  ;swap to X
1681
1682                         ;Not super fast, but smaller?? than a LDD 0,y, STD 0,x twice? NOPE 11 vs 8
1683                         ;it IS universal for size changes though.
1684                         ;;         ldy #Curmbf           ;2 dest, active rules in ram
1685                         ;;         ldab BFCNT            ;3 # to xfer
1686
1687                         ;we don't have to worry about ints here since SFLG should stop recursion
1688                         ;;         jsr COPYM             ;3 copy to ram (domode)
1689  FB45   BD FC 76                            jsr copybf         ;;save redundant loads by jumping into COPYR
1690  FB48   7E FA 4E                            jmp FORCER         ;3 make it fire
1691
1692                         ;;         LDD 0,x               ;2 get data (8)
1693                         ;;         STD Curmbf            ;2 put data
1694                         ;;         LDD 1,x               ;2 2nd word
```

```
                  ::        STD Curmbf+2        ;2 put

                  ;****
                  ;Digital output only possible in single chip mode (supposedly)
                  ;although we could use unused port A bits
FB4B              digou:
                  :         ldaa 1,x  ;get output pattern
FB4B  F7 10 04                        stab DIGOUT         ;output pattern to defined port
FB4E  7C 00 1B              dcont:    inc Outadr+1        ;point to real output for gtimer
                  ;@@ bug if rollover occurs in low address byte if >256 EE for rules
FB51  20 10                           bra gtimer          ;digo

                  ;*****
                  dosubs:   ;call a subroutine & setup timer
FB53  16                    tab                 ;move value to useful place
                  ;         ldaa #'s'
                  ;         bsr dbg             ;s

FB54  C0 F8                           subb #$F8           ;subtract base
FB56  58                    aslb                ;*2 for addresses
FB57
FB57  CE F8 4F                        ldx #SUBADR         ;get the base
FB5A  3A                              abx                 ;index in
FB5B  EE 00                           ldx 0,x             ;get the address into X
FB5D  AD 00                           jsr 0,x             ;call it

FB5F  20 02                           bra gtimer          ;standard finish

FB61              domot:
FB61  8D 13                           bsr LD_MOTOR        ;stuff timer
                  ; then fall into gtimer
FB63              gtimer:
FB63  CE F8 10                        ldx #RULTIM         ;timer bit fields
FB66  8D 1A                           bsr BFSCN           ;check timer bitfields

FB68  24 0B                           bcc ?timdn          ;no timer for this rule.
                                                ;C unchanged with DECA in BFSCN
FB6A  DE 1A                           ldx Outadr          ;recover pointer to output value
FB6C  EC 01                           ldd 1,x             ;get rule # & timer
FB6E  2A 03                           bpl ?settm          ;positive rule #... forced timer

                  ;@@       anda #$7F           ;mask off high bit
FB70  DD 1E                           std Inhrul          ;neg rule#, inhibit timer
FB72  39                    rts
FB73              ?settm:
FB73  DD 1C                           std Frcrul          ;save here for timint

                  ;         ldaa #'t'  ;dbg
                  ;         jmp dbg             ;t

FB75  39          ?timdn:   rts                 ;done for now
                            .page
```

```
                ;*****
                ; loc_motor: lock motor in safe postion
                ;*****
                ;loc_motor:                  ;used by EEUDATE
                ;         ldaa MSAFE         ;get safe positon from EE
                ; fall into LD_MOTOR
                ;*****
                ; LD_MOTOR. Scale motor value in !!A!! to 16 bit timer value Mtime
                ;*****
FB76            LD_MOTOR:
                ;         cmpa Latmot        ;check previous setting (Last Motor Time)
                ;         beq ldr            ;skip computes if same (for speed)
FB76
                ;         staa Latmot        ;save motor time for next time in
FB76  40                           tsta              ;;set CCs
FB77  27 03                        beq zero  ;no bias since zero requested
FB79  BB F8 04                     adda MBIAS        ;add in bias from EE

                ;@@@ should be EE? so dynamic scaling possible
FB7C  C6 14               zero:    ldab #MDEG        ;degrees -> time
FB7E  3D                           mul               ;compute it
FB7F  DD 70                        std Mtime         ;and stuff here for output ints
FB81  39        ldr:      rts
                          .page
```

```
                ;****
                ; BFSCN. checks a bit field for a 1 at the rule # in B
                ; and returns Carry Set if high
                ; X points to BF, uses Scnrul for rule to check
                ; BFSCNB. B contains rule # being checked
                ; Alters A
FB82            BFSCN:
                ;         pshb               ;save
FB82  D6 20                        ldab Scnrul       ;rule being checked
                ;BFSCNB:
FB84  17                  tba                ;copy
FB85  54                  lsrb               ;into byte offset, no sign extension
FB86  54                  lsrb
FB87  54                  lsrb

FB88  3A                           abx               ;index into bit field list
FB89  E6 00                        ldab 0,x  ;get bits
FB8B  84 07                        anda #$7          ;mask A to bit # in byte

FB8D  58                  shlb:    lslb              ;into carry
FB8E  4A                           deca              ;count down, C unaffected
FB8F  2C FC                        bge shlb  ;until done 8 times

                ;         pulb               ;recover
FB91  39                  rts                ;return with Carry set/clr from chosen bit
                          .page
```

```
                              ;*** DVT code
                              ; FIFO(16),A/D(8),DVT(8). Fifo runs toward lower addresses.
                              ;0   $8   $10   $18
FB92                          DERIVE:
                              ;          bsr fclr   ;dbg flag

FB92  37                                 pshb               ;save B
FB93  CE 00 50                                 ldx #Fifo ;base
FB96  C4 07                                    andb #$7        ;mask to channels only
FB98  3A                                       abx             ;offset to channel #

FB99  A6 08                                    ldaa $8,x       ;get f(1) =previous
FB9B  C6 04                                    ldab #4
FB9D  3D                                       mul             ;4*f(1)
FB9E  DD 49                                    std Dtemp       ;save for later use

FBA0  A6 10                                    ldaa $10,x      ;f(0)
FBA2  C6 03                                    ldab #3
FBA4  3D                                       mul             ;3*f(0)

FBA5  93 49                                    subd Dtemp      ;3*f(0)-4*F(1)
FBA7  DD 49                                    std Dtemp       ;save again

FBA9  E6 00                                    ldab $0,x       ;f(2)
FBAB  4F                                       clra            ;high order clear
FBAC  D3 49                                    addd Dtemp      ;3*f(0)-4*F(1)+f(2)

                              ;by way of explanation for the following code
                              ;DECI  HEX
                              ;385  0181 after ASR shift values
                              ;129  0081
                              ;128  0080 if msbyte is zero before shift then data will be ok, else pmax
                              ;127  007F
                              ;001  0001
                              ;000  0000
                              ;-01  ffff
                              ;-127 ff81
                              ;-128 ff80
                              ;-129 ff7f if msbyte is FF before shift then data will be ok, else nmax
                              ;-385 fe7f
FBAE  4D                                       tsta            ;check high order info
FBAF  27 0E                                    beq ?bok        ;plus normal, exit ok

FBB1  2B 04                                    bmi ?ftst ;check negative

FBB3  C6 7F                                    ldab #127       ;positive limit
FBB5  20 0A                                    bra ?advt

FBB7  81 FF                         ?ftst:     cmpa #$FF       ;other limit
FBB9  27 04                                    beq ?bok        ;neg normal, exit ok

FBBB  C6 80                                    ldab #-128      ;neg limit
FBBD  20 02                                    bra ?advt       ;store it

FBBF  47                      ?bok:     asra             ;/2
FBC0  56                                       rorb            ;the long way since no ASRD instruction

FBC1  E7 18                         ?advt:     stab $18,x      ;store here. DVT data above FIFO & A/D data
FBC3  33                                       pulb            ;recover channels
                              ;         bsr fset  ;clear debug flag
FBC4
FBC4  39                                       rts             ;all done
                              ;18 bytes*

                                        .page
```

```
                     ;**** Sign-on message
                     ;SIGNON:        db CR,LF,"03/28/91"," "+$80        ;say hello

                     ;**************************
                     ;****** MAIN PROGRAM ******
                     ;**************************
                     ;certain OPTION, INIT & TMSK2 bits can only be written once!
                     ;up to 64 E cycles after reset, so we MUST configure
                     ;here immediately after power-up or COP reset.
                     START:  ;once only at power up
FBC5  CE 10 00                        ldx #REG           ;register base
                     ;          ldaa #$01         ;Ram @ $0, Reg @ $1000
                     ;          staa $103D        ;INIT @ reset. But RESET put it this way ANYWAY

FBC8  86 92                           ldaa #$92          ;for options...
                     ;ADPU=1=on CSEL=0=E IRQE=0=Lvl DLY=1=D CME=0=off x=0 CR=11=cop=.26 Sec
FBCA  A7 39                           staa _OPTION,X     ;make any changes now!

                     ;          ldaa #$00         ;Block Protect off for now
                     ;          staa BPROT        ;
FBCC  6F 35                           clr _BPROT,X

                     ;          ldaa #$00         ;timer base rate .5uSec @ 2Mhz, all ints off
                     ;          staa TMSK2        ;set rate now!
FBCE  6F 24                           clr _TMSK2,X

                     ;This really should be done by EVM board since a change to CONFIG requires
                     ;that ALL EEprom in -A1 be erased (Bulk is only way to erase config)
                     ;High 4 bits determin EEPROM location in -A2 chip (2K on upper 1/2 of 4k bound
            )
                     ; if! expanded, else $F800 if single chip mode.
                     ; EE overlays ROM in -A2 Single chip therefore no Buffalo routines access!
                     ;          ldaa #$ED         ; $E for EE in -A2, noseo=1 nocop=1 romon=0 eeon=1
                     ;          staa CONFIG       ;set config! But since EE cell don't have to.

                     ;          ifdef ILLBUG      ;doing debug on illegal instructions traps
                     ;                            ;usually cause by bad rule structures
                     ;          ldd #0000
FBD0  4F                        clra
FBD1  5F                        clrb              ;the small way to get D=0
FBD2  DD 44                           std Mebb           ;prevent trash first time around
FBD4  20 0C                           bra BEGIN          ;skip this next bit

                     ;we end up here on an illegal instruction or SWI trap
FBD6  30                        ISTART: tsx               ;get old stack for illegal instruction trap
FBD7  DF 44                           stx Mebb           ;save for output later
FBD9  18 CE 00 AA                     ldy #IN_BUFSTA     ;dest
FBDD  C6 10                           ldab #16 ;this many
FBDF  BD FC 7D                        jsr COPYM          ;copied (istart) & PRAY stack is OK!??
                     ;          endif
                     ;*****
                     ;normal restart of program (^R)
                     ;*****
FBE2                 BEGIN:
FBE2  0F                        sei               ;mask off int's if on
FBE3  8E 00 FF                        lds #TOS           ;load stack at top of ram

```

```
FBE6  8D 4D                              bsr INITS          ;initialize sub-systems

FBE8  0E                          cli             ;clear int mask. Int's enabled globally
                          ;@@@ and other stuff yet

                          ;* signon
                          :         ldx #SIGNON       ;Say hello
                          ;         jsr PMSG
                          ;SIGNON text is only version number since we've run out of space
                          ;         ldaa #VER          ;current version
                          :         jsr HOUTS          ;output

FBE9  CE F8 00                           ldx #DATE          ;Rule date in EE
FBEC  C6 03                              ldab #3            ;this many
FBEE  BD FE 88                           jsr PHMSG          ;saved in hex

                          :         ifdef ILLBUG
FBF1  DE 44                              ldx Mebb           ;crashed?
FBF3  27 0D                              beq ?ncrsh         ;no

FBF5  96 20                              ldaa Scnrul        ;last scanned rule
FBF7  BD FE AF                           jsr HOUTS          ;inform

FBFA  C6 10                              ldab #16 ;this many out
FBFC  CE 00 AA                           ldx #IN_BUFSTA     ;from here!
FBFF  BD FE 88                           jsr PHMSG          ;inform user
FC02                      ?ncrsh:
                          ;         endif

                          ;* power on start command
                          ;         ldaa lstcmd        ;last entered command saved in eeprom
                          ::        ldaa #'7' ;starting cmd
                          ;         jsr OUT_QC         ;echo

                          ::        bra TSTCMD         ;fake it

                          ;********
                          ;MAIN program command loop
                          ;********
FC02  BD FE 60                 lpin:     jsr lprompt        ;new line & rule # on screen
FC05  12 41 08 F9              lpin:     brset Flag2,CFLG,lpin        ;Rule changed? print it

                          ;         wai            ;Wait for any interrupt (SCI desired)
FC09  BD FF 21                           jsr IN_DQ          ;Get char?

FC0C  25 F7                              bcs lpin  ;Wait until we have one

FC0E  13 40 40 09                        brclr Flag1,EFLG,necho       ;Echo? none if clr

FC12  81 0D                              cmpa #CR
FC14  26 02                              bne echo

FC16  20 EA                              bra lpin  ;crlf & prompt

FC18  BD FE 3E                 echo:     jsr OUT_QC         ;Echo it

FC1B                      necho:
```

```
FC1B                TSTCMD:
FC1B   CE F9 00             ?cmds:  ldx #CMDTBL       ;Use this table
FC1E   BD F9 32                     jsr CMDDISP       ;CoMmanD DISPatch
FC21   25 02                        bcs ?err3         ;Command not found

FC23   20 DD                        bra lpin   ;Get next command

FC25   86 21                ?err3:  ldaa #'!'  ;Bad command response
FC27   BD FE 3E                     jsr OUT_QC        ;Send it
FC2A   BD FE 3E                     jsr OUT_QC        ;Twice

FC2D   D6 A4                ?ewt:   ldab Out_bffil    ;Has everything been printed?
FC2F   26 FC                        bne ?ewt          ;Hang around until it is

FC31   20 CF                        bra lpin   ;Start over

FC33   0C                   NULL:   clc               ;stop errors
FC34   39                           rts               ;Null cmd input characters jsr to here

                            .page
```

```
                          ;**** subsystem initialize ****
FC35                      INITS:
                          ;@@ could do this with a LDD, SDD if space gets tight
                          ;          ldaa #EFLG          ;start with...
                          ;no XOFFR, ECHO, , , , , , no HEX INPUT
                          ;          staa Flag1          ;stuff it

                          ;          ldaa #0             ;nothing here for now
                          ;          staa Flag2
FC35  CC 40 00                              ldd #((EFLG)*256+0)     ;combined, Flag1*256 +Flag2
FC38  DD 40                                 std Flag1          ;stuffed here
FC3A
FC3A  DD 42                                 std Flag3          ;rule Flags3 starts out CLEAR (Flag4 too)

FC3C  CC 00 40                              ldd #Tmot          ;dummy address initially
FC3F  DD 48                                 std Adjadr         ;stuffed here for tune

FC41  F6 F8 05                              ldab MSAFE         ;an initial pulse width from EE
FC44  D7 4D                                 stab Tmot          ;to protect motor.

FC46  8D 41                                 bsr SCIINI         ;starup sci, X will point to REG base on exit

      [01]                       ifndef CHIPA2
                                 jsr ERUDATE       ;update EE if required
      [00]                       endif

                          ;      ldx #Filblk       ;block to fill
                          ;      ldd #$0010        ;Fill block, length
                          ;?flp: sta 0,x           ;stuff fill value
                          ;      inx
                          ;      decb              ;count down
                          ;      bne ?flp  ;until done (256 max)
                          ;12 bytes

                          ;      ldd #0000         ;zero
FC48  4F                         clra              ;Zeros
FC49  5F                         clrb              ;the small way to get D=0
FC4A  DD 10                             std Srcadr         ;so no errors on startup
FC4C  DD 70                             std Mtime          ;no pulses at start
FC4E  DD 18                             std Stime          ;no stop time
                          ;#     std Btime         ;no false beeper

                          ;@ maybe Currul = $FF batter?
FC50  5A                                decb               ;turn into rule into $FF
FC51  DD 18                             std Curmod         ;start with MODE00 (A) & this rule (B) on power up

                          ;! this is where the startup rule is fired
FC53  CC 01 01                          ldd #$0101         ;force rule 1 on 1st timint (##tt)
FC56  DD 1C                             std Frcrul         ;by stuffing here

                          ;      ldd #$0001        ;# no repeat, .2 beep on reset
FC58  DD 3D                             std Btime          ;# beep .2 sec on startup

FC5A  CE 10 00                          ldx #REG           ;get base again (EEUDATE may corrupt)
                          ;      ldaa #$00         ;set all bits as input
                          ;;     ldaa #$FF         ;set all bits as output for now (no pullups)
                          ;      staa DDRC         ;Data DiR C
```

```
FC50   6F 07                              clr _DDRC,X         ;all inputs on C

FC5F   86 83                              ldaa #$83           ;PA1 is out. RTInt @ 32.77mS if enabled
FC61   A7 26                              staa _PACTL,X       ;config A ports bit & Real Time Int rate

                     ;spiini:
FC63   C6 FF                              ldab     #$FF       ;nearly all bits output (*SS is gen purpose out)
FC65   E7 09                              stab _DDRD,X        ;set port D direction

FC67   C6 51                              ldab     #%01010001      ;SPIE.SPE.DPSH/PUL.MSTR  CPOLH.CPHA0.SPR1&0
FC69   E7 28                              stab _SPCR,X        ;Enable SPI as Master @ E/4 baud

                     ;          ldab _SPSR,X       ;Fake a read to clear SPIF flag
                     ;          ldab _SPDR,X       ;so writes possible

                     ;******
                     ;startup timer OC1
                     ;we won't do this here since we don't want a restart to be visible, eh?
                     ;1st OC1 int 32msec after power up??
                     ;          ldd TCNT           ;currently
                     ;          addd #RTCRAT       ;some time from now
                     ;          std TOC1           ;setup

FC6B   86 80                              ldaa #$80           ;OC1 bit
                     ;          staa _OC1M,X       ;let OC1 affect OC1 bit
                     ;          staa _OC1D,X       ;set OC1 high on compares, others low
                     ;done earlier         staa _PACTL,X       ;enable output driver OC1
FC6D   A7 22                              staa _TMSK1,X       ;enable OC1 timer ints ($80)

                     ;@@@ this should be under interrupt (and maybe power down if slow enough)
                     ; Start A/D scanning/converting
FC6F   86 30                              ldaa #$30           ;CCF/x/scan/mult ch0-4 :scan 0-4 repeatedly
                     ;          ldaa #$10          ;scan 0-4 once
FC71   A7 30                              staa _ADCTL,X       ;start A/D

                     ;*** COPYR copies MODE00 to ram
FC73   CE F8 16                 COPYR:    ldx #MODE00         ;src here
FC76   18 CE 00 12              copybf:   ldy #Currnbf        ;dest  here
FC7A   F6 F8 0B                           ldab BFCNT          ;bit field size
                     ;fall into
                     ;****
                     ;COPYM. X points to Src, Y to Dest, B is size. Called everywhere
                     ;****
FC7D   A6 00                    COPYM:    ldaa 0,x   ;get data
FC7F   18 A7 00                           staa 0,y   ;stuff here
FC82   08                                 inx                 ;bump up pointers
FC83   18 08                              iny
FC85   5A                                 decb                ;count down
FC86   2E F5                              bgt COPYM           ;until done

FC88   39                                 rts
                     ;**** end of inits

                     ;****
                     ;SCIINI. Initialize HDPTR, TLPTR, BFSIZ , BAUD, SCCR's
                     ;****
                     ;could use copym but only one byte saved 18L vs 17M
```

```
FC89  CC 00 AA        SCIINI:  ldd #IN_BUFSTA   ;init hdptrs & tlptrs to start of buffers
FC8C  DD A6                    std In_hdptr
FC8E  DD A8                    std In_tlptr

FC90  CC 00 80                 ldd #OUT_BFSTA
FC93  DD A0                    std Out_hdptr
FC95  DD A2                    std Out_tlptr

                  ;     ldd #0000        ;buffers start out empty
FC97  4F                clra             ;buffers start out empty
FC98  5F                clrb             ;the small way to get D=0
FC99  DD A4                    std Out_bffil      ; which clears In_bffil in following byte too

                  ; Initialize SCI registers
FC9B  CE 10 00                 ldx #REG         ;register base
                  ;     ldaa #$00        ;1/8/1 bits. Idle line wakes rcvr
                  ;     staa SCCR1
FC9E  6F 2C                    clr _SCCR1,X     ;setup

                  ;EE byte for baud rate
                  ;     ifndef CHIPA2
                  ;     ldaa $F803       ;from here
                  ;     else
FCA0  B6 F8 03                 ldaa BDRATE      ;$30=E/13/1>9600 $34=E/13/16>600 baud
                  ;     endif

                  ;     staa BAUD
FCA3  A7 2B                    staa _BAUD,X

                  SCIIEN:      ;EEUDATE/SAVE returns here to restart Rxint's
FCA5  CE 10 00                 ldx #REG         ;base again because of EE (6 vs 8)
FCA8  1C 2D 2C                 bset _SCCR2,X,$2C      ;RcvrIntE,TxEn,RxEn
FCAB  39                       rts
                  ;**** end of sciini
                        .page
```

```
                          ;****
                          ; EE UpDATEs only changed bytes since faster than whole rewrite
                          ; Copies a piece of code to RAM at IN_BUFSTA since EE disappears on EELAT
                          ;****
                          ;@@@ we now only update a small block at a time
                          ;svmsg:  db "avin","g"+$80

                          ;**** SAVE! udates EE
FCAC                      SAVE:
                          ;        bsr EEUDATE       ;called at command level now
                          ;        bra SCIIEN        ;restart Rxints
FCAC                      EEUDATE:
                          ;        ldx #svmsg        ;inform
                          ;        jsr PMSG          ;user

                          ;@@ we should be so fast that no one notices 10msec hick-up
                          ;        sei               ;turn off ints & pray for no COP, XIRQ

                          ;@ strictly speaking RXint ONLY needs to be off at this point. Later ALL off.
FCAC  CE 10 2D                       ldx #SCCR2       ;register to mess with
FCAF  1D 00 20                       bclr 0,x,$20     ;kill rxint enable (6 bytes)

FCB2  CE FD 00                       ldx #pbee        ;src
FCB5  18 CE 00 AA                    ldy #IN_BUFSTA   ;dst. since usually empty
FCB9  C6 10                          ldab #(pbeee-pbee)    ;length less than 256! (16 actually)

FCBB  BD FC 7D                       jsr COPYM        ;copy EE subroutine to ram

FCBE  DE 10                          ldx Srcadr       ;current source in ram (EE write adr)
                          ;        beq pbret         ;nothing to save, AN ERROR
FCC0  8C F8 00                       cpx #EE          ;where EE sits
FCC3  25 31                          blo pbret ;not EE, skip it!

FCC5  18 CE 00 00                    ldy #Wrkrul      ;ram table
FCC9  C6 10                          ldab #RSIZ       ;is this big
FCCB  18 A6 00            pcomp:     ldaa 0,y  ;get current ram data
FCCE  A1 00                          cmpa 0,x         ;compare with previous saved EE
FCD0  27 1E                          beq bdec         ;same so skip pgming this byte
                          ;
FCD2  37                             pshb             ;save counter
FCD3  18 3C                          pshy             ;save current pointer
                          ;dbg...
FCD5  36                             psha             ;save data
FCD6  BD FE A8                       jsr HOUTC2       ;dbg

FCD9  32                             pula             ;get back
FCDA  36                             psha             ;save again
FCDB  BD FE AF                       jsr HOUTS        ;dbg

FCDE  32                             pula             ;get data back
                          ;...dbg
FCDF  C6 16                          ldab #$16        ;byte erase first
FCE1  8D 15                          bsr pbe          ;do it

FCE3  0E                  cli              ;allow an int (but no rxint) since rules still in Ram
FCE4  81 FF                          cmpa #$FF        ;did we only need an erase?
FCE6  27 04                          beq ffdone       ;yes
```

```

FCE8   C6 02                                   ldab #$2 ;program
FCEA   8D 0C                                   bsr pbe             ;now

FCEC   0E                        ffdone: cli                       ;allow a few more ints
FCED   18 38                                   puly                ;recover em
FCEF   33                        pulb

FCF0   08                bdec:   inx                    ;bump pointers ahead
FCF1   18 08                                   iny
FCF3   5A                                      decb                ;and count down
FCF4   26 D5                                   bne pcomp           ;until zero

FCF6                     pbret:
FCF6   20 AD                                   bra SCIIEN          ;restart Rxints
                         ;       rts                    ;done update

     [01]                        ifdef COPON            ;copy enabled
                         ;copset resets the COP timer
                         copset: ldab #$55              ;COP needs attention
                                 stab COPRST
                                 comb                   ;flip it to $AA, one byte shorter
                                 stab COPRST
                                 rts

                         ;**** code compression subr
                         pbe:    bsr copset             ;reset cop

     [01]                        else
FCF8                     pbe:
     [00]                        endif

FCF8   18 CE 08 98                             ldy #2200           ;2500=10 msec@ 4cy/dey 3333=10msec@3/dex  +3/bne
FCFC   0F                        sei                    ;stop ALL ints & prey no XIRQ
                         ;       jmp pbee               ;program byte/erase & ret
FCFD   7E 00 AA                                jmp IN_BUFSTA       ;its really here

                         ;***the following 16 bytes get copied to ram so EEUDATE can call with
                         ;B set to erase or pgm cmd, Y with delay value, and X pointing to address,
                         ;A with value to pgm
FD00                     pbee:
FD00   F7 10 3B                                stab PPROG          ;set EELAT
FD03   A7 00                                   staa 0,x  ;write or erase @ x
FD05   7C 10 3B                                inc PPROG           ;EEPGM up
FD08   18 09                     wt10:         dey                 ;count down
FD0A   26 FC                                   bne wt10            ;until done
FD0C   7F 10 3B                                clr PPROG           ;finished
FD0F   39                        rts                    ;return to EE code
     FD10                pbeee   equ $                  ;end of part copied to ram. 16 bytes
                         ;**** end of EEUDATE
                                 .page
```

```
                        ;*****
                        ; ENTER| Fiddles memory locations. Operates directly if ADDR < $B000
                        ; ADJUST| Adjust values of rule table. Asks for rule #, copies to ram if not
                        ; already there, and then asks for parameter to change.
                        ;@ have fun...
                        ;*****
                        ;avmsg:  db "nter ","@" + $80
                        ;rnmsg:  db "dj ","#" + $80
FD10  00 0A C5                   elmsg:  db CR,LF,"E" + $80        ;"Elem","#" + $80
                        ;nvmsg:  db "Ne","w" + $80
                        ;****
FD13                    update:
FD13  9C 10                              cpx Srcadr       ;same as current request in X?
FD15  27 0D                              beq ?endud       ;yes, no RAM update required

                        ;@ do we want to prompt for save? Do we want a reread?
                        ;        jsr EEUDATE     ;different, need to update EE before overwrite
FD17
FD17  DE 44                              ldx Msbb         ;recover requested source address
FD19  DF 10                              stx Srcadr       ;new set will be from here
FD1B  18 CE 00 00                        ldy #Wrkrul      ;destination is..
FD1F  C6 10                              ldab #RSIZ       ;this big
FD21  BD FC 7D                           jsr COPYM        ;copy new set to ram

FD24  DE 10                      ?endud: ldx Srcadr       ;get it again (unfortunately)
FD26  DF 25                              stx Offadr       ;fix offset for print outs
FD28  CE 00 00                           ldx #Wrkrul      ;this is where data now (or still) sits
FD2B  DF 4B                              stx Adjadr       ;setup work pointer
FD2D  39                                 rts

                        ;**** ENTER|      ;uses absolute address, no rule fiddling
FD2E                    ENTER:
                        ;        ldx #evmsg      ;enter value message
                        ;        bsr GETNUM      ;input an address. Returns to main if nothing entered
FD2E  8D 67                              bsr GETVAL       ;get a value. Return to main if none.

FD30  CE 01 10                           ldx #$0110       ;header 1 line, 16 elements
FD33  BD FE D6                           jsr ENTDMP       ;do header & data @ Msbb

FD36  4F                         clra            ;zero
FD37  5F                         clrb            ;the small way to get D=0
FD38  DD 25                              std Offadr       ;assume direct addr. Clear offset

FD3A  DE 44                              ldx Msbb         ;get the address to fiddle
FD3C  8C B6 00                           cpx #$B600       ;lower limit of EE
FD3F  25 2E                              blo fuladr       ;no EE udate, use it directly

FD41  8D D0                              bsr update       ;update EE if different from current

FD43  20 2A                              bra fuladr       ;to common code

                        ;**** ADJUST| twiddle rules, friendlier user interface
FD45                    ADJUST:
                        ;        ldx #rnmsg      ;rule # msg
                        ;        bsr GETNUM      ;input rule #. Returns to main if nothing entered
FD45  8D 50                              bsr GETVAL       ;get a value. Return to main if none.

```

```
2309  FD47  D6 45                     ldab Lsbb          ;get rule # into B
2310  FD49  37                        psnb               ;save for 2nd rule difference
2311  FD4A  BD FA 3A                  jsr GETEDR         ;get rule's address from indexed table (we think)
2312                  ;         bcs ?nrul         ;already in RAM
2313
2314                  ;         stx Adjadr        ;save it here for later use
2315  FD4D  DF 44                     stx Msbb           ;;& stuff for memdmp to use
2316  FD4F  8D C2                     bsr update         ;update EE if needed
2317
2318  FD51  33                ?nrul:  pulb               ;recover rule #
2319  FD52  5C                        incb               ;next rule #
2320  FD53  BD FA 3A                  jsr GETEDR         ;get next rules's address
2321                  ;bcs not needed here since if rule was in ram, this one wouldn't be
2322                  ;and if it wasn't, rule would have been put there by update
2323
2324  FD56  8F                xgdx              ;into D
2325  FD57  93 10                     subd Srcadr        ;find address difference
2326                  ;         bne ?adnz         ;address difference NOT zero, may be ok
2327
2328                  ;if difference=0 then same rule at both locations. Do we get the NEXT one up &

2329                  ;try again?? For now, just limit to 16 elements.
2330  FD59  C4 0F                     andb #$0F          ;and B has length (16 max)
2331  FD5B  26 02                     bne ?bnz           ;reasonable length?
2332
2333  FD5D  C6 10                     ldab #$10          ;limit to 16
2334  FD5F  86 01             ?bnz:   ldaa #1            ;one line of info, b has length
2335  FD61  8F                xgdx              ;swap to X
2336  FD62  BD FE CB                  jsr ADJDMP         ;;let memdump do the work
2337
2338  FD65  CE FD 10                  ldx #elmsg         ;element #
2339  FD68  8D 2A                     bsr GETNUM         ;get info, returns to main if no entry
2340
2341                  ;         ldd Adjadr        ;recover basic address
2342                  ;         addd Msbb         ;include element offset (16bit)
2343                  ;         xgdx              ;into x
2344                  ;since we can't be more than 16 elements offset
2345  FD6A  DE 48                     ldx Adjadr         ;get basic address in RAM!!
2346  FD6C  D6 45                     ldab Lsbb          ;include element offset
2347  FD6E  3A                        abx                ;include in X
2348  FD6F  8D 12             fuladr: bsr prnadr         ;tell them
2349
2350  FD71  20 3C                     bra tadj  ;tune it
2351
2352                  ;RTDSPH uses these. X preset to Adjadr by CMDSPH
2353  FD73            SETADR:
2354  FD73  BD FF 3D                  jsr HEX_BIG        ;get an address
2355  FD76  25 26                     bcs getret"        ;no input, use old value (RTS)
2356
2357  FD78  4F                clra
2358  FD79  5F                clrb              ;the small way to get D=0
2359  FD7A  DD 25                     std Offadr         ;clear offsets
2360
2361  FD7C  DE 44                     ldx Msbb           ;get input value
2362  FD7E  20 03                     bra prnadr         ;stuff & print
2363
2364  FD80            INCADR:
```

```
FD80  08                          inx              ;modify pointer
FD81  08                          inx              ;compensate for falling into dex
FD82
FD82               DECADR:
FD82  09                          dex              ;modify pointer

                   ;limit the address to modulo 16
                   ;        xgdx          ;swap to D for limits
                   ;        clra          ;high order of address <256 (RAM @ 0)
                   ;        andb #$F      ;modulo 16. A will be zero
                   ;        xgdx          ;back to X
                   ;fall into ...

FD83  DF 4B             prnadr:   stx Adjadr       ;save for RTDSPH functions
FD85  3C                          pshx             ;save Ram address
FD86  8F                    xgdx          ;swap into D
FD87  D3 25                       addd Offadr      ;equivalent address offset (since data's at zero now)
FD89  8F                    xgdx          ;back to X

FD8A  BD FE A8                    jsr HOUTC2       ;print it

FD8D  38                          pulx             ;recover ram address
FD8E  BD FE A2                    jsr CRLF  ;new line &

FD91  7E FF 0C                    jmp svp          ;prompt for adjustments

                   ;*****
                   ;GETNUM: prompts with message @ X & gets a number with echo. Returns to main i

                   ;no entry.
                   ;GETVAL As above but no message.
                   ; Placed here so BSR's in possible
FD94  BD FE 71              GETNUM:       jsr PMSG         ;prompt user

FD97  BD FF 3D              GETVAL:       jsr HEX_BIQ      ;get value

                   ;        bcs NULL      ;no entry, return directly
FD9A  24 01                       bcc ?getok       ;no error, normal return

FD9C  38                          pulx             ;pop callers address. Returning to MAIN          now
FD9D               ?getok:
                   ;        ldx Mabb      ;get value
                   ;        jmp HOUT2S    ;echo it

FD9D  0C                          clc              ;make certain no error to MAIN
FD9E  39                    getret:  rts      ;also used by SETADR

                   ;*****
                   ; TUNE| write values to the motor
                   ;@ Could we use adjust functions??
                   ;*****
                   ;tnmsg:  db "une"," " + $80
FD9F               TUNE:
                   ;        ldx #tnmsg
                   ;        jsr PMSG      ;tune msg

FD9F  14 41 01                    bset Flag2,TFLG  ;stop state diagram, set tuning flag
```

```

FDA2  CE FD C7                          ldx #texit          ;tune exit
FDA5  3C                                pshx                ;setup as return for RTDSPH if ESC entered

                          ::      ldx #Adjadr       ;last address adjusted
                          ::      ldaa 0,x  ;get value
                          ::      staa Tmot         ;save here

FDA6  CE 00 4D                          ldx #Tmot           .address to fiddle
FDA9  DF 4B                             stx Adjadr          ;setup for keyboard

FDAB  4F                                clra
FDAC  5F                                clrb                ;the small way to get D=0
FDAD  DD 25                             std Offadr          ;Zero offset for correct print out

                          ;entry here from enter and adjust
FDAF  BD FE A2                  tadj:   jsr CRLF  ;new line for echos
FDB2  BD FF 21                  ?slp:   jsr IN_DQ           ;get a char
FDB5  25 FB                             bcs ?slp  ;none avail, loop here

FDB7  BD F9 2F                          jsr RTDSPH          ;adjust a variable
FDBA  25 F6                             bcs ?slp  ;not one of ours so loop again

FDBC  13 41 01 F2                       brclr Flag2,TFLG,?slp         ;not tuning motor, skip motor update code

FDC0  96 4D                             ldaa Tmot           ;from new temporary value
FDC2  BD FB 76                          jsr LD_MOTOR        ;scale to Mtime
FDC5
FDC5  20 EB                             bra ?slp  ;and hang around until ESC causes exit

                          ; entered by escexit's attempt to return to main
FDC7                      texit:
FDC7  15 41 01                          bclr Flag2,TFLG     ;reenable state machine
                          ; adjust's abort exit too
FDCA                      endadj:
FDCA  39                                rts                 ;return to main
                                  .page
```

```
                        ;***********
                        : Serial communication interrupt routine
                        ;***********
                        :; Need to enable ints earlier. Clear rxint & txint for each routines section
                        .; also need a get char function that doesn't move tlpntr so editing possible!

                        :@ Needs to be sped up. @9600B less than 1msec/char available, + other ints!
FDC8                    SCIINT:
FDC8  CE 10 00                      ldx #REG          ;base
                        ;     !daa SCSR         ;who interrupted?
FDCE  A6 2E                         ldaa _SCSR,x      ;who interrupted?
FDD0  85 20                         bita #$20         ;Tx/Tc/Rdrf/Idl  Or/Nf/Fe/X
FDD2  27 36                         beq trentst       ;is NOT the receiver, maybe tx

                        ;     ldaa SCDR         ;IS receiver, so get character
FDD4  A6 2F                         ldaa _SCDR,x      ;IS receiver, so get character
FDD6  84 7F                         anda #$7f         ;clear parity

FDD8  81 13                         cmpa #XOFF        ;is it XOFF?
FDDA  26 05                         bne ?xont         ;nope

FDDC  14 40 80                      bset Flag1,XFRFLG         ;stop further tx. Xoff received
FDDF  20 52                         bra notran        ;stop immediately

FDE1  81 11             ?xont:      cmpa #XON         ;:XON?
FDE3  26 0B                         bne wt_in         ;nope, must EN_Q it

FDE5  15 40 80                      bclr Flag1,XFRFLG ;XON, let tx go
FDE8
FDE8  96 A4                         ldaa Out_bffil    ;stuff avail? (lda shorter than tst)
FDEA  27 3F                         beq endint        ;no, so fini, nothing to TX now that we're allowed

FDEC  8D 41                         bsr txon  ;startup tx ints

FDEE  20 1A                         bra trantst       ;and process pending int

                        ;****
                        ; Queue the character in acca --> input buffer
                        :; change to 8 bit offsets for speed/size? ABX with b as offset
                        ;****
                        ;@ this is a TRAP. Can't get characters out since we're hung waiting
                        ;@ in an ISR!!!
FDF0  D6 A5             wt_in:      ldab In_bffil     ;get current size
FDF2  C1 11                         cmpb #IN_BFMAX ;check for room

                        ;* send XOFF if near full

                        ;     bhi wt_in         ;we'll have to wait for room
FDF4  22 14                         bhi trantst       ;we'll just have to loose char while we wait for room
                        :; could load bfmax into b and loop on cmp bffil for tighter loop
                        :@could also allow 2 chars more into buffer until xoff takes effect

FDF6  3C                            pshx              ;save reg pointer
FDF7  DE A6                         ldx In_hdptr      ;get head pointer
FDF9  A7 00                         staa 0,X  ;and store char in que

FDFB  7C 00 A5                      inc In_bffil      ;one more in queue, safe from Int's
```

```

FDFE  08                    inx                     ;step pointer ahead
FDFF  8C 00 8F                          cpx #IN_BFEND    ;past end?
FE02  23 03                             bls in_norap1    ;no, no wrap needed
FE04  CE 00 AA                          ldx #IN_BUFSTA  ;yes, wrap back to the start of buffer

FE07                  in_norap1:
FE07  DF A6                             stx in_hdptr      ;update hdptr
FE09  38                    pulx                    ;recover reg pointer

                      ;fall through to..

                      ;****
                      ;txint serviced here after rx does work (also rx bra's if XON/XOFF)
                      ;****
FE0A                  trantst:
FE0A  A6 2E                             ldaa _SCSR,X      ;is it the transmitter?
FE0C  2A 1D                             bpl endint        ;no, so we're done poll (TXBE=MSB, high is neg)

FE0E  12 40 80 21                       brset Flag1,XFRFLG,notran   ;Xoff? Yes, turn off txint for now

FE12  96 A4                             ldaa Out_bffil    ;is there anything left to transmit?
FE14  27 1D                             beq notran        ;empty, so skip to end & kill future ints

FE16  DE A2                             ldx Out_tlptr     ;get pointer
FE18  A6 00                             ldaa 0,x  ;get char
FE1A  B7 10 2F                          staa SCDR         ;send it out
FE1D
FE1D  08                                inx               ;skip to next spot
FE1E  8C 00 9F                          cpx #OUT_BFEND ;did we fall off end of buffer?
FE21  23 03                             bls out_nowrp2    ;no, skip wrap

FE23  CE 00 80                          ldx #OUT_BFSTA  ;yes, point back to the start

FE26                  out_nowrp2:
FE26  DF A2                             stx Out_tlptr     ;update tail pointer
FE28  7A 00 A4                          dec Out_bffil     ;one less occupied space

                      ;;        cmpa #OUT_BFMIN ;should we send XON character?
                      ;;        bhi out_nxon       ;no

                      ;;Send XON,  spaca available
                      ;;out_nxon:

FE2B  3B              endint:   rti                ;return later

                      ;**** called by rxint for Xon, and OUT_QC :> A  must be preserved
FE2C  CE 10 00                  txonx:  ldx #REG          ;for out_qc
FE2F  1C 2D 80                  txon:   bset _SCCR2,x,$80          ;set Tx Int En if not already up

FE32  39                        rts

                      notran:   ;@ bclr??          ;entered by rxint XOFF
                      ;         ldaa SCCR2         ;get current flags
                      ;         anda #$7f          ;disable transmit interrupts
                      ;         staa SCCR2         ;by lowering enable

```

```
                          :        ldx #REG          :pointer
FE33  1D 2D 80                             bclr _SCCR2,x,$80      :disable tx int's

                          XRET:    ;xirq rti
FE36  3B                           rti               :all done here

                          ;***** here for bsr limits
FE37  0D 0A 20 20 20               spcmsg: db CR,LF,"    ","  "+$80     :blanks for memdmp
  FE3C  20 A0
                          ;*****
                          ; Queue the character in acca  -> output buffer
                          ; Returns A unchanged
                          ;;; change to 8 bit offsets from x to improve speed/size since only 256 RAM
                          ;;; use 8 more !
                          ;Preserves B & X & A
                          ;*****
FE3E                      OUT_QC:
FE3E  37                           pshb              ;Save bytes elsewhere
FE3F  3C                           pshx              ;since we're doing this a lot

FE40  D6 A4                        wt_out:  ldab Out_bffil     ;get current size

                          ;@@@ should probably be a ram location
FE42  C1 1F                                 cmpb #OUT_BFMAX          ;check for room
FE44  22 FA                                 bhi wt_out          ;we'll have to wait for room
                          ;; or load ac with max and loop on compare bffil for tighter loop
                          ;;perhaps set carry and return instead of looping

FE46  DE A0                                 ldx Out_hdptr       ;get head pointer
FE48  A7 00                                 staa 0,X ;and store char in que

FE4A  7C 00 A4                              inc Out_bffil       ;one more in queue ;Int's can't affect inca

FE4D  08                                    inx                 ;step  pointer ahead

FE4E  8C 00 9F                              cpx #OUT_BFEND ;past end?
FE51  23 03                                 bls out_norap1      ;no, no wrap needed

FE53  CE 00 80                              ldx #OUT_BFSTA  ;yes, wrap back to the start of buffer

FE56                      out_norap1:
FE56  DF A0                                 stx Out_hdptr       ;update hdptr

FE58  8D D2                                 bsr txonx           ;start up ints now that X can be mangled

FE5A  38                                    pulx                ;get it all back
FE5B  33                           pulb
FE5C  0C                                    clc                 ;prevent errors

FE5D  39                                    rts                 ;of OUT_QC
                                   .page
```

```
                    ;****;
                    ;FREMOD:        bclr Flag2,MFLG   ;clear mode locked flag
                    ;          ldx #?frmsg       ;and inform
                    ;          bra mend          ;common end

                    ;?frmsg:  db "re","e" + $80
                    ;lcmsg:   db "ocke","d" + $80

                    ;****|
                    ;LOCMOD:        bset Flag2,MFLG   ;set mode locked flag
                    ;          ldx #lcmsg
                    ;mend:     ;fall into PMSG
                    ;          bra PMSG
                    ;**** some of the commands called from RTTBL
                    ; Esc exit. CMDDISP jumps to here if ESC entered
FE5E                ESCEXIT:
                    ;          pulx              ;pop RTDSPH return (2 not used if BRA cmdsph)
FE5E  38                       pulx              ;pop caller return (Tune, Dump, etc)

FE5F  39            rtend:     rts               ;should land us in main

                    ;*****
                    ;puts up new line prompt
FE60                lprompt:
FE60  8D 40                         bsr CRLF          ;New line on screen

FE62  96 18                         ldaa Curmod       ;get current mode
FE64  8D 59                         bsr OUTRH         ;right hand nybble out

FE66  96 19                         ldaa Currul       ;current new rule
FE68  8D 45                         bsr HOUTS         ;print rule

FE6A  15 41 08                      bclr Flag2,CFLG   ;clear current rule needs print flag

FE6D  86 2A                         LDAA #PROMPT      ;Prompt character
FE6F  20 CD                         bra OUT_QC        ;Stuff in que & return

                    ;***** end lprompt

                    ;****
                    ;PMSG. print string @ X, null or hi-bit terminated
                    ;BSRs OUT_QC
                    ;****
FE71                PMSG:
                    ;;         tpa               ;get CC
                    ;;         psha              ;save. others may need C bit info

FE71  A6 00                         ldaa 0,x  ;get char
FE73  27 09                         beq ?pmd          ;zero is end

FE75  84 7F                         anda #$7F         ;clear high bit to get rid of funny IBM chars
FE77  8D C5                         bsr OUT_QC        ;print it (PMSG)

FE79  6D 00                         tst 0,x           ;check for end, MSB high
FE7B  08                   inx               ;point to next byte
FE7C  2A F3                         bpl PMSG          ;more if MSB=0
FE7E                ?pmd:
```

```
                  :         clc                 ;stop possible errors (tst cleared C)

                  ::        pula
                  ::        tap                 ;restore CC's
                  ;@CC save/restore currently bombs. WHY?

FE7E   39                   rts                 ;all done

                  ;**** CR then CNTRO
                  ;CRCNTR:         bsr CRLF              ;newline

                  ;**** counts up # of values in B, printing as it goes
                  ;CNTRO: clra                  ;start @ zero
FE7F              CNTR:
                  ::        pshx                ;save
FE7F   37                   pshb                ;this too
FE80              cntrl:
                  :         psha                ;save
                  :         bsr HOUTS           ;print it

                  :         decb                ;count down
                  :         bitb #7             ;modulo 7
                  :         bne ?nspc2          ;no space yet

                  :         bsr OUTS            ;cntr spc out

                  ;?nspc2: pula                 ;recover
                  :         inca                ;next value

                  :         tstb                ;flags
                  :         bne cntrl ;not yet
                  ; 5 bytes saved. (looked like 10 but bar/rts/pshx/pulx ate 5)
FE80   8D OF                          bsr phbc ;subr to print, countb etc, saves a & incs

FE82   26 FC                          bne cntrl ;B not zero, more to do

FE84   33                   pulb                ;recover
                  ::        pulx                ;;too
FE85   39                   rts

                  ;****
                  ;PHMSG. print hex string @ X, space separated, B has length
                  ;BSRs HOUTS
                  ;Saves B
                  ;****
FE86   8D 1A                PHMSGC:             bsr CRLF          ;new line
FE88   37         PHMSG: pshb                   ;save it
FE89   A6 00                ?phmsg: ldaa 0,x    ;get value
FE8B   8D 04                          bsr phbc ;call print subr

FE8D   26 FA                          bne ?phmsg        ;not done yet

FE8F   33                   pulb                ;recover
FE90   39                   rts                 ;return

                  ;**subr for CNTR & PHMSG
FE91   36         phbc:     psha                ;save for other caller (cntr)
```

```
FE92  8D 1B                              bsr HOUTS         ;hex out with space (PHM)

FE94  08                        inx               ;next up (or wasted cycles if cntr)
FE95  5A                                 decb              ;count down
                    ;           bitb #$7 ;modulo 7
FE96  C5 03                              bitb #$3 ;modulo 4
FE98  26 02                              bne ?nospc        ;no addition space output

FE9A  8D 15                              bsr OUTS          ;add an extra one

FE9C  32                        ?nospc:  pula              ;recover. the other caller (cntr) needs this
FE9D  4C                                 inca              ;count up.

FE9E  5D                        tstb              ;set flags
FE9F  39                        rts               ;return with CCs set

                    ;**** CRLF sends out cr & lf
                    ;Leaves A as LF
                    ;**** CRR CR only
                    ;**** HOUTSC. HOUTS + CRLF
FEA0  8D 0D                     HOUTSC:           bsr HOUTS         ;print it first
                    ;fall into
FEA2  8D 71                     CRLF:    bsr CRR           ;CR first

FEA4  86 0A                              ldaa #LF ;LF next
FEA6  20 96                     outa:    bra OUT_QC        ;stuff & return (also used by HOUT.. exits)

                    ;****
                    ; HOUT2S. Hex OUT 2 Space from X value
                    ; A returns as SPC. X. B unchanged
                    ; BSRs: HOUT, HOUTS
FEA8  8D F8                     HOUTC2:           bsr CRLF          ;newline
FEAA                HOUT2S:
FEAA  3C                                 pshx              ;save current value
FEAB  32                                 pula              ;get high order byte
FEAC  8D 07                              bsr HOUT          ;print it (HO2S)

FEAE  32                        pula              ;get low order byte
                    ;           bsr HOUTS         ;print it with space
                    ;fall into
                    ;****
                    ; Hex OUT Space. Hex out with space
                    ; A returned as space, B & X ok
                    ; JSRs: OUT_QC
                    ; BSRs:  HOUT
                    ;****
FEAF                HOUTS:
FEAF  8D 04                              bsr HOUT          ;print it

FEB1  86 20                     OUTS:    ldaa #SPC         ;followed by..
                    ;           bra OUT_QC        ;which returns
FEB3  20 F1                              bra outa ;which returns

                    ;****
                    ; Hex OUT. converts the byte in A to hex and outputs
                    ; Calls: OUT_QC.
                    ; Destroys A. B & X ok
```

```
                      ;****
      FEB5   36       HOUT:   psha             ;save it
      FEB6   8D 03                    bsr ?outlh          ;send left half

      FEB8   32               pula             ;recover
      FEB9   20 04                    bra OUTRH           ;send right half and let IT return

      FEBB   44       ?outlh: lsra             ;get high nybble first
      FEBC   44                       lsra
      FEBD   44                       lsra
      FEBE   44               lsra

      FEBF   84 0F            OUTRH:  anda #$0f           ;mask out high nybble
      FEC1   8B 30                    adda #$30           ;convert to numeric
      FEC3   81 39                    cmpa #$39           ;in 0-9 range?
      FEC5   2F DF                    ble outa  ;yes

      FEC7   8B 07                    adda #$07           ;bias to A-F
      FEC9   20 DB                    bra oute  ;send & return (hout)
                      ;****


                      ;*****
                      ; MEMDMP[ dumps 256 Memory values at address given
                      ; Bsr's to HOUT & HOUTS & HOUTC2 &
                      ; ENTDMP uses Msbb for address, lowX for width, hiX for line count.
                      ;*****
                      ;llth      equ 16              ;line length memdmp
      FECB   4F       ADJDMP:         clra                ;start header at zero always
      FECC   20 0A                    bra memin

      FECE   BD FF 3D         MEMDMP:         jsr HEX_BIQ        ;get start value    ;@@ bsr?
      FED1   25 23                    bcs mend            ;no entry so quit.

      FED3   CE 10 10                 ldx #$1010          ;16 lines, 16 elements

      FED6   96 45            ENTDMP:         ldaa Lsbb          ;start header count here
      FED8   97 48            memin:  staa Acnt           ;stuff for later use
      FEDA   DF 46                    stx Lcnt  ;set line count & length from X

      FEDC   CE FE 37                 ldx #spcmsg         ;first part
      FEDF   8D 90                    bsr PMSG            ;of header (MDMP)

      FEE1   96 48                    ldaa Acnt           ;get starting count
      FEE3   D6 47                    ldab Llth  ;& line length
      FEE5   8D 98                    bsr CNTR            ;the rest of the header

      FEE7   D6 46                    ldab Lcnt           ;# of lines to dump
      FEE9   DE 44                    ldx Msbb            ;load fetch pointer

      FEEB   8D BB            ?mlp:   bsr HOUTC2          ;print CR, X & space

      FEED   37               pshb             ;;save line counter

      FEEE   D6 47                    ldab Llth ;;this many out
      FEF0   8D 96                    bsr PHMSG           ;;of this string

      FEF2   33               pulb             ;;recover line counter
```

```
FEF3  5A                    decb              ;count loop down
FEF4  26 F5                        bne ?mlp          ;on we go

FEF6  39          mend:     rts               ;until done # of lines

                  ;***** INC/DEC var addresses
                  ;X set by RTDSPH to Adjadr
FEF7              INCVAR:
FEF7  6C 00                        inc 0,x           ;bump up value
FEF9  20 11                        bra svp

FEFB              DECVAR:
FEFB  6A 00                        dec 0,x
FEFD  20 0D                        bra svp

FEFF              SETVAR:
FEFF  BD FF 3D                     jsr HEX_BIQ       ;get a value
FF02  25 F2                        bcs mend          ;no input, use old value (RTS)

FF04  D6 45                        ldab Lsbb         ;lower order of value input
FF06  DE 4B                        ldx Adjadr        ;recover value's address
FF08  E7 00                        stab 0,x  ;change value

                  ;         bsr CRLF          ;overwrite old screen data
FF0A  8D 09                        bsr CRR           ;overwrite old screen data

FF0C  E6 00                 svp:   ldab 0,x  ;retrieve new value
FF0E  86 7C                        ldaa #'|' ;delimiter
                  ;****
                  ; print character in A & value in B with trailing space
FF10              PCHARV:
FF10  8D 05                        bsr outc  ;bsr OUT_QC      ;print the char (pcv)

FF12  17                    tba               ;get value
                  ;         bra HOUTS         ;dump out and return
FF13  8D 9A                        bsr HOUTS         ;dump out

                  ;         bra CRR           ;cr only & return
                  ;fall into

FF15  86 0D                 CRR:   ldaa #CR          ;CR only
FF17  7E FE 3E              outc:  jmp OUT_QC        ;stuff & return (CRR)

                  ;*** end of PCHARV & CRR

                  ;******
                  ; PRULS| prints formated rules to screen (eventually) using adjust subs
                  ; Redundant!
                  ;PRULS:                     ;dbg
                  ;         ldx #$0110        ;header 1 line, 16 elements
                  ;         jsr ENTDMP        ;do header & data @ Msbb

                  ;         ldx Adjadr
                  ;         bsr HOUTC2

                  ;         ldx Msbb
                  ;         bra HOUTC2


                         ;******** end of PPARS
                                 .page
```

```
                        ;****
                        ; IN_DQ. DeQueue an input character into A
                        ; IN_EDQ. Echoed DeQueue
                        ; Alters A
                        ; Use B ? Indexed refs? common OUT_DQ code?
                        ;****
                        ;@ use in_dqw for those callers looping with BCS to IN_DQ

FF1A   8D 05                     IN_EDQ:  bsr IN_DQ          ;get a char

FF1C   25 08                              bcs nie            ;no input to echo so end, passing C

                        ;;       bsr IN_DQW       ;wait for a char
FF1E   7E FE 3E                           jmp OUT_QC         ;echo it & return

                        ;;IN_DQW:         bsr IN_DQ          ;call it
                        ;;       bcs IN_DQW       ;hang around

                        ;;       rts

FF21                    IN_DQ:
FF21   96 A5                              ldaa in_bffil      ;get size (TST larger)
FF23   26 02                              bne in_cont        ;have data?

FF25   0D                        sec               ;NO, set carry & return A as zero
FF26   39               nie:     rts

FF27                    in_cont:
FF27   3C                        pshx              ;save it now
FF28   DE A8                              ldx in_tlptr       ;get pointer
                        ;        ldab 0,x ;get char
FF2A   A6 00                              ldaa 0,x ;get char
FF2C
FF2C   08                        inx               ;skip to next spot
FF2D   8C 00 BF                           cpx #IN_BFEND      ;did we fall off end of buffer?
FF30   23 03                              bls ?in_nowrp2     ;no, skip wrap

FF32   CE 00 AA                           ldx #IN_BUFSTA     ;yes, point back to the start
FF35                    ?in_nowrp2:
FF35   DF A8                              stx in_tlptr       ;update tail pointer
FF37   7A 00 A5                           dec in_bffil       ;one less occupied. Int's can't split this instruction


                        ;; should ldaa bfmin and compare directly?
                        ;        ldaa in_bffil     ;
                        ;        cmpa #IN_BFMIN   ;should we send XON character?
                        ;        bpl in_nxon      ;no

                        ;***** Yes, send XON,  space available
                        ;@@@ this needs development. How do you jam into output stream? Use Flags?
                        ;in_nxon:
                        ;        tba              ;copy to a
FF3A   0C                                 clc                ;no error flag
FF3B   38                        pulx              ;recover reg
FF3C   39                        rts               ;we're done

                        ;*****
```

```
                    ; HEX_BIN. Convert Ascii hex value
                    ; DEC_BIN. Convert the characters in buffer to decimal and save in msbb/lsbb
                    ; Stops at non-numeric char which gets returned in upper case.
                    ; Carry set if no data to do
                    ; Mangles A,B, X
                    ;*****
                    ;@@@ Should NOT convert char to upper case??
FF3D  86 3F                 HEX_BIQ:ldaa #'?' ;prompt...
FF3F  BD FE 3E                      jsr OUT_QC        ;for input (hexbiq)

FF42                HEX_BIN:
                    ;       bset Flag1,HFLG   ;set hex bit
                    ;       bra ?cldd         ;skip past decimal

                    ;DEC_BIN:
                    ;       bclr Flag1,HFLG   ;clr hex bit

                    ;@@@ may want a form that skips clear?
FF42  4F            ?cldd:  clra              ;clear value
FF43  5F                    clrb              ;the smaller way
FF44  DD 44                         std Msbb          ;here, gets Lsbb too
                    ;       bclr Flag1,NFLG   ;start out positive

FF46  8D 18                         bsr rgchk *       ;get char & range check

FF48  25 34                         bcs cend          ;;out of bounds, return error, no input

FF4A  20 04                         bra ?a2           ;;ok, lets go with 1st time in
                    ;       bcc ?a2           ;a valid 1st char
                    ;       cmpa #'-'         ;negative sign?
                    ;       beq ?sneg         ;set negative

                    ;       cmpa #'+'         ;plus
                    ;       bne cend          ;invalid 1st char
                    ;       bclr Flag1,NFLG   ;clr negative flag
                    ;       bra ?a1
                    ;?sneg: bset Flag1,NFLG   ;set negative flag
                                              ;fall into additional chars loop
FF4C  8D 12                 ?a1:    bsr rgchk         ;get char & check limits

FF4E  25 24                         bcs aend          ;not valid, we're done
                    ;       bcs anend         ;not valid, we're done
FF50
FF50  80 30                 ?a2:    suba #'0'         ;OK so far, strip down ascii
FF52  16                    tab               ;place in low half
FF53  8F                    xgdx              ;;swap to x

FF54  DC 44                         ldd Msbb          ;get old value
FF56  05                    asld              ; *2
FF57  05                    asld              ; *2=4
                    ;       brclr Flag1,HFLG,ddec   ;hex way?

FF58  05                    asld              ;yes, *2=8
                    ;       bra ?dhex         ;do hex

                    ;?ddec: addd Msbb         ; +1=5
                    ;?dhex:
```

```
FF59  05                   asld                  ; *2=X10 or X16

FF5A  8F                   xgdx                  ;;swap back
                                   abx                  ;;add in digit
FF5C  DF 44                        stx Msbb             ;;store it
FF5E  20 EC                        bra ?a1              ;loop for more
                   ;*****
                   ;Range check subr. Get character and check for numeric (or hex)
FF60  8D B8                rgchk:  bsr IN_EDQ           ;get char with echo
FF62  25 FC                        bcs rgchk            ;none avail, wait

FF64  81 60                        cmpa #'''            ;lower case?
FF66  2D 02                        blt ?rgck ;no, less than 1st L.C. char

FF68  80 20                        suba #$20            ;change to upper case
FF6A               ?rgck:
                   ;       brclr Flag1,HFLG,dchk        ;Hex accepted? Bra if not

FF6A  81 41                        cmpa #'A'            ;in range?
FF6C  2D 08                        blt dchk  ;too small, may be 0-9

FF6E  81 46                        cmpa #'F'
FF70  2E 0C                        bgt cend  ;too big to be used

FF72  80 07                        suba #7              ;shift it to 10-15
                   ;       bra aend  ;ok exit

FF74  0C           aend:   clc                  ;return no error code & next char in upper case
FF75  39                   rts                  ;we're done. Return to caller

                   ;anend: brclr Flag1,NFLG,aend         ;not negative, do normal exit
                   ;       ldd Msbb             ;get result
                   ;       coma
                   ;       comb
                   ;       addd #1              ;two's compliment negation
                   ;       bra aend  ;clean exit

FF76  81 30                dchk:   cmpa #'0'            ;in range?
FF78  2D 04                        blt cend  ;too small

FF7A  81 39                        cmpa #'9'
                   ;       bgt cend ;too big, set carry
FF7C  2F F6                        ble aend  ;ok ending

FF7E  0D           cend:   sec                  ;return error code & next char in UC
FF7F  39           hend:   rts
                   ;**** end of hex_bin etc
                   *****
                   ; DUMP| print the state, CH 0-7 & Digin
                   ;
                   ;*****
FF80               DUMP:
FF80  BD FE 60                     jsr Iprompt          ;fresh line & prompt with Currul & "*"

FF83  C6 07                        ldab #7              ;starting here
FF85  BD FB 92             ?dvs:   jsr DERIVE           ;compute the derivative data
```

```
3079  FF88  5A                              decb            ;count down
3080  FF89  2C FA                           bge ?dvs        ;until done all
3081
3082  FF8B  CE 00 60                        ldx #Anldat     ;Data stored here every OC1 int
3083  FF8E  C6 10                           ldab #16 ;length of string
3084  FF90  BD FE 88                        jsr PHMSG       ;print in hex (dump)
3085
3086  FF93  B6 10 03                        ldaa PORTC      ;current digital values
3087  FF96  8D 3A                           bsr outb  ;jsr HOUTS         ;out + spc (dump)
3088
3089                      ;using RTC for delay
3090  FF98  C6 04                           ldab #SDRATE    ;get rate value
3091  FF9A  CE 10 00                        ldx #REG        ;get the base
3092  FF9D  1D 25 BF                ?wtrto: bclr _TFLG2,x,$BF ;R/M/W knock down RTIF ($40)
3093  FFA0                ?cklp:
3094  FFA0  BD FF 21                        jsr IN_DQ       ;any keys? (dump)
3095
3096  FFA3  24 09                           bcc ?srt  ;yes, do em
3097
3098  FFA5  1F 25 40 F7                     brclr _TFLG2,x,$40,?cklp    ;hang around until RTIF high
3099
3100  FFA9  5A                              decb            ;count out time
3101  FFAA  2E F1                           bgt ?wtrto      ;hang around some more
3102
3103  FFAC  20 D2                           bra DUMP        ;another time around
3104
3105                      ;?srt:    jsr RTDSPH      ;check for escape or tune
3106  FFAE  81 1B                   ?srt:   cmpa #ESC       ;escape
3107  FFB0  26 CE                           bne DUMP
3108
3109  FFB2  39            dend:     rts
3110
3111                      ;******
3112                      ;FLAGS| toggles various flag bits
3113  FFB3  8D 88                   FLAGS:  bsr HEX_BIQ     ;get bit #
3114  FFB5  25 FB                           bcs dend        ;no value, skip out
3115
3116  FFB7  CE 00 40                        ldx #Flag1      ;start here
3117                      ;         ldab Lsbb       ;get value
3118                      ;;        cmpb #16        ;upper limit
3119                      ;;        bhi dend ;too big (this covers negative too since unsigned)
3120                      ;         bmi dend        ;negative, so skip
3121
3122                      ;         cmpb #8         ;threshold for Flag2
3123                      ;         bit ?flg1  ;use as is
3124
3125                      ;         inx             ;point to Flag2
3126                      ;         cmpb #16        ;threshold for Flag3
3127                      ;         bit ?flg1  ;ok
3128
3129                      ;         inx             ;point to Flag3
3130                      ;?flg1:
3131                      ;18 ^ vs 12 v
3132  FFBA  96 45                           ldaa Lsbb       ;get value
3133  FFBC  16                      tab             ;copy
3134  FFBD  56                      rorb            ;get highest 5 bits in position
3135  FFBE  56                      rorb
```

```
3136  FFBF   56                 rorb
3137  FFC0   C1 04                     cmpb #4         ;upper limit with 4 flags
3138  FFC2   22 EE                     bhi dend  ;too big so skip out
3139
3140  FFC4   3A                        abx             ;index into Flags table
3141
3142  FFC5   16                 tab               ;copy again
3143  FFC6   C4 07                     andb #$7        ;mask to remainder modulo 8
3144  FFC8   4F                 clra              ;start empty
3145  FFC9   0D                        sec             ;make a high bit
3146
3147  FFCA   49                 ?alp:  rola            ;shift across
3148  FFCB   5A                        decb            ;count it
3149  FFCC   2C FC                     bge ?alp  ;until -1 (base 0)
3150
3151  FFCE   A8 00                     eora 0,x  ;flip the bits into a
3152  FFD0   A7 00                     staa 0,x  ;and put back
3153  FFD2   7E FE AF           outb:  jmp HOUTS       ;echo result & return (lprompt)
3154
3155
3156                     ;*****
3157                     ;TEST1| tests whatever
3158                     ;*****
3159                     ;TEST1:
3160                     :         inc SPDR        ;bump up spi data
3161                     :         ldy #100        ;preload this value
3162                     :         jsr wt10  ;wait a bit
3163
3164                     :         ldx Flag1       ;getem both (DATA)
3165                     :         jsr HOUTC2      ;dumpem
3166
3167                     :         ldx #SPCR       ;SPI
3168                     :         ldab #3         ;length, prints all three SPI regs
3169                     :         jmp PHMSG       ;message
3170  FFD5
3171                     :         ldaa Andat      ;get ch0 value
3172                     :         jmp LD_MOTOR    ;use it here
3173
3174                     :         jsr HEX_BIQ     ;get a value
3175                     :         ldaa Msbb
3176                     :         bsr HOUTS       ;upper
3177                     :         ldaa Lsbb
3178                     :         jmp HOUT        ;print it and return
3179                               .page


3180                     ;******
3181                     ; HELP| print out screen of help info
3182                     ;******
3183                     ;HELP:   ldx #HLPMSG     ;Get pointer
3184                     :       jmp PMSG        ;Print help string & return
3185
3186                     ;HLPMSG:        db CR,LF
3187                     :       db "? Adj Dmp Entr Mdmp Tun Sav ^Rst ^Tst",CR,LF+$80
3188                             .page
```

```
3189                      ;****************
3190                      ;Interrupt Vectors
3191                      ;****************
3192 FFD5                         ABSOLUTE        ;absolute positions
3193                      ;       org $FFD2       ;reserved but we'll use the space
3194                      ;RSTT:   jmp START      ;unknow ints trap to here and cause restart
3195                      ;****
3196 FFD6                         org $FFD6
3197 FFD6  FDCB                           dw SCIINT       ;FFD6 SCI int
3198 FFD8  FBC5                           dw START        ;FFD8 SPI int
3199 FFDA  FBC5                           dw START        ;FFDA PA1
3200 FFDC  FBC5                           dw START        ;FFDC PAOV
3201 FFDE  FBC5                           dw START        ;FFDE TOI
3202
3203                      ;       org $FFE0
3204 FFE0  FBC5                           dw START        ;FFE0 OC5 int
3205 FFE2  FBC5                           dw START        ;FFE2 OC4
3206 FFE4  FBC5                           dw START        ;FFE4 OC3
3207 FFE6  FBC5                           dw START        ;FFE6 OC2
3208 FFE8  F947                           dw TIMINT       ;FFE8 OC1
3209 FFEA  FBC5                           dw START        ;FFEA IC3
3210 FFEC  FBC5                           dw START        ;FFEC IC2
3211 FFEE  FBC5                           dw START        ;FFEE IC1
3212
3213                      ;       org $FFF0
3214 FFF0  FBC5                           dw START        ;FFF0 Real Time Int
3215 FFF2  FBC5                           dw START        ;FFF2 Interrupt ReQuest
3216 FFF4  FE36                           dw XRET             ;FFF4 XIRQ quick RTI
3217 FFF6  FBD6                           dw ISTART       ;FFF6 SoftWare Int
3218 FFF8  FBD6                           dw ISTART       ;FFF8 Illegal INStruction
3219 FFFA  FBC5                           dw START        ;FFFA Computer Operating Properly watchdog
3220 FFFC  FBC5                           dw START        ;FFFC CLocK Monitor
3221 FFFE  FBC5                           dw START        ;FFFE power on RESET
3222
3223 0000                         END             ;of real code


          Lines Assembled : 3223        Assembly Errors : 0
```

Claims

1. In an above knee prosthesis (AKP) having upper and lower leg segments and a connecting knee joint, the improvement comprising:

   a linear, hydraulic damper for separately and variably damping each of flexion and extension rotational movements of the knee joint;

   electronic sensing means for measuring each of AKP knee angle and lower leg segment strain and emitting signals indicative thereof;

   actuating means for adjusting the damper to vary damping of the knee joint in flexion and extension; and

   programmed computer means for receiving the emitted signals from the sensing means and comparing them to stored threshold values which are indicative of pre-determined transition points selected for adjustment of at least one of flexion and extension damping, and, when the received signal values correlate with stored values, causing the actuating means to vary damping.

2. A method for controlling the knee joint of an above knee prosthesis having a knee joint, lower leg and ankle, comprising:

   storing, in a computer memory, threshold values of lower leg strain and knee angle, which values are indicative of the knee joint bending in stance phase, of anterior positioning of the center of gravity of body weight relative to the ankle, and of swing phase, all in the course of a step along a level surface;

   continuously sensing lower leg strain and knee angle during use of the prosthesis and producing electronic signals corresponding thereto;

comparing the signals against the stored threshold values and, when the signals substantially correlate with threshold values, automatically altering the rate of rotation of the knee joint in one or both of flexion and extension, as required.

3. A bi-directional variable linear hydraulic damper for use in an above knee prosthesis, comprising:

a hollow closed cylinder comprising end walls and a side wall forming a chamber for retaining hydraulic fluid, each end wall forming a rod opening;

a cylindrical hollow piston disposed in the cylinder chamber and adapted to slide longitudinally therein, said piston having axial rods extending through the rod openings in sealed engagement with the cylinder;

said piston carrying an exterior circumferential seal ring between its ends, said seal ring being in sealing relationship with the cylinder side wall, said piston being formed by end walls and a side wall, said piston forming a first aperture through its wall above the seal ring and a second aperture through its wall below the seal ring;

a first one way check valve controlling the first aperture for enabling ingress of fluid into the piston chamber from the first end of the cylinder chamber;

a second one way check valve controlling the second aperture for enabling ingress of fluid into the piston chamber from the second end of the cylinder chamber;

a first pair of diametrically opposed ports extending through the piston side wall adjacent its first end, on one side of the seal ring;

a second pair of diametrically opposed ports extending through the piston side wall adjacent its second end, on the other side of the seal ring; and

valve means for progressively reducing or increasing the effective area available for fluid flow of the first ports and separately progressively reducing or increasing the effective area available for fluid flow of the second ports.

4. In an above knee prosthesis (AKP) for use by a human user, said AKP having upper and lower leg segments, a knee joint connecting the segments, and a foot attached to the base of the lower leg segment, the improvement comprising:

means, pivotally connected with the leg segments, for separately and variably damping each of flexion and extension rotational movements of the knee joint;

electronic sensing means for monitoring AKP knee angle and position of the center of gravity of the user's body relative to the AKP foot and emitting signals indicative thereof;

actuating means for adjusting the damping means to vary damping of the knee joint; and

programmed computer means for receiving the emitted signals from the sensing means and continuously establishing from said signals the state of the AKP in the course of a movement and activating the actuating means to vary damping to substantially simulate natural knee action.

5. The improvement as set forth in claim 4 wherein the damping means comprises:

a pair of closed chambers for containing hydraulic fluid,

means, connected to the leg segments and forming two passageways connecting the chambers, for moving fluid from one chamber to the other through one of the passageways when the leg segments are moving together and through the other of the passageways when the leg segments are moving apart, and

means for regulating the flow of fluid through each passageway;

said actuating means being adapted to adjust the regulating means to vary damping of the knee joint.

6. The improvement as set forth in claim 4 wherein the damping means is a bi-directional variable linear hydraulic damper comprising:

a hollow closed cylinder comprising end walls and a side wall forming a chamber for retaining hydraulic fluid;

a cylindrical hollow piston disposed in the cylinder chamber and adapted to slide longitudinally therein;

said piston carrying an exterior circumferential seal ring between its ends, said seal ring being in sealing relationship with the cylinder side wall, said piston being formed by end walls and a side wall, said piston forming a first aperture through its wall above the seal ring and a second aperture through its wall below the seal ring, said piston dividing the cylinder chamber into closed first and second end chambers;

a first one way check valve controlling the first aperture for enabling ingress of fluid into the piston chamber from the first end chamber;

a second one way check valve controlling the second aperture for enabling ingress of fluid into the piston chamber from the second end chamber;

a first pair of diametrically opposed ports extending through the piston side wall adjacent its first end, on one side of the seal ring;

a second pair of diametrically opposed ports extending through the piston side wall adjacent its second end, on the other side of the seal ring; and

valve means for progressively reducing or increasing the effective area available for fluid flow of the first ports and separately progressively reducing or increasing the effective area available for fluid flow of the second ports;

said actuating means being adapted to adjust the valve means to vary damping of the knee joint.

7. The improvement as set forth in claim 4 wherein the programmed computer means is adapted to compare the emitted signals against stored threshold values indicative of transition points between states of a repetitive movement of the AKP and, when the signals substantially correlate with threshold values, to alter the rate of rotation of the knee joint in one of or both of flexion and extension.

8. The improvement as set forth in claim 7 wherein the stored threshold values are selected from the group consisting of the absolute and derivative values of knee angle and the position of the center of gravity of the user's body relative to the AKP foot, the duration from the last transition point and the possible future states in the course of the movement.

9. The improvement as set forth in claim 8 wherein:

the sensing means for monitoring the position of the center of gravity of the user's body relative to the AKP foot consists of means for monitoring lower leg strain.

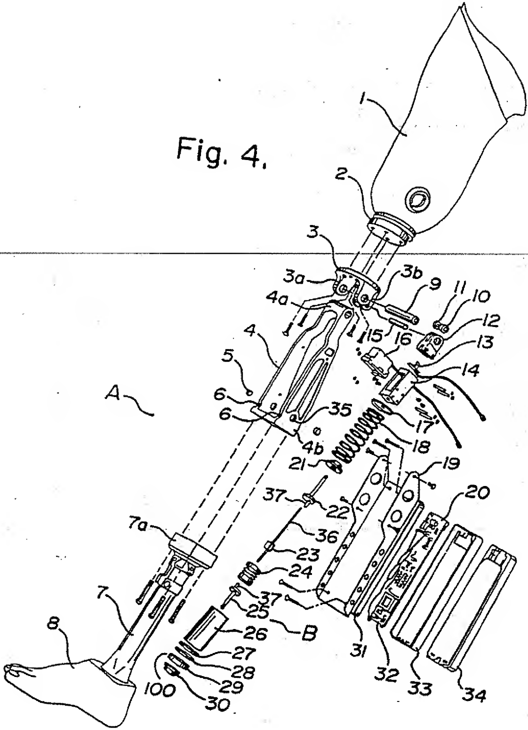
10. The improvement as set forth in claim 6 wherein the programmed computer means is adapted to compare the emitted signals against stored threshold values indicative of transition points between states of a repetitive movement of the AKP and, when the signals substantially correlate with threshold values, to alter the rate of rotation of the knee joint in one of or both of flexion and extension.

11. The improvement as set forth in claim 10 wherein the stored threshold values are selected from the group consisting of the absolute and derivative values of knee angle and the position of the center of gravity of the user's body relative to the AKP foot, the duration from the last transition point and the possible future states in the course of the movement.

12. The improvement as set forth in claim 11 wherein:

the sensing means for monitoring the position of the center of gravity of the user's body relative to the AKP foot consists of means for monitoring lower leg strain.

Fig. 4.


1
2
3
3a
3b
4a
4
5
6
6
4b
35
9
11
10
12
13
14
15
16
17
18
19
20
21
37
22
36
23
24
37
25
26
27
28
29
30
100
B
31
32
33
34
7a
7
8
A

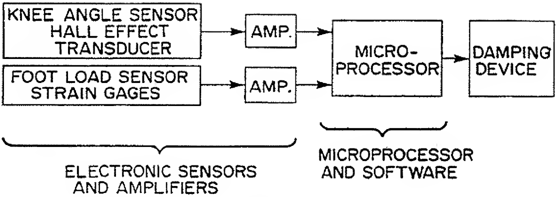

KNEE ANGLE SENSOR HALL EFFECT TRANSDUCER
FOOT LOAD SENSOR STRAIN GAGES
AMP.
AMP.
MICRO-PROCESSOR
DAMPING DEVICE
ELECTRONIC SENSORS AND AMPLIFIERS
MICROPROCESSOR AND SOFTWARE

Fig. 1.

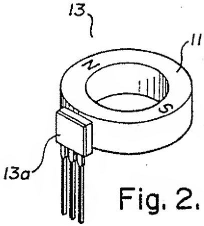


Fig. 2.

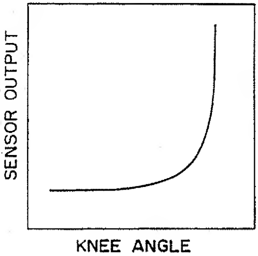


Fig. 3.

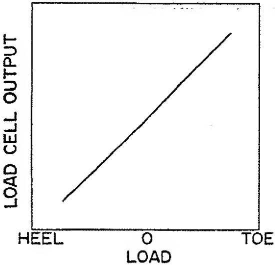
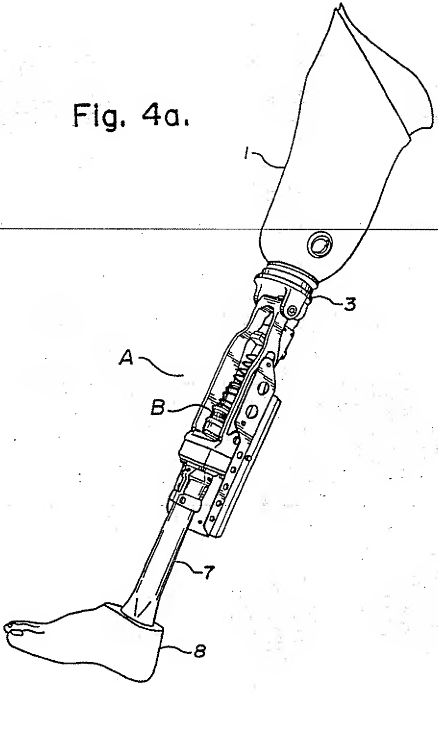


Fig. 5.

Fig. 4a.



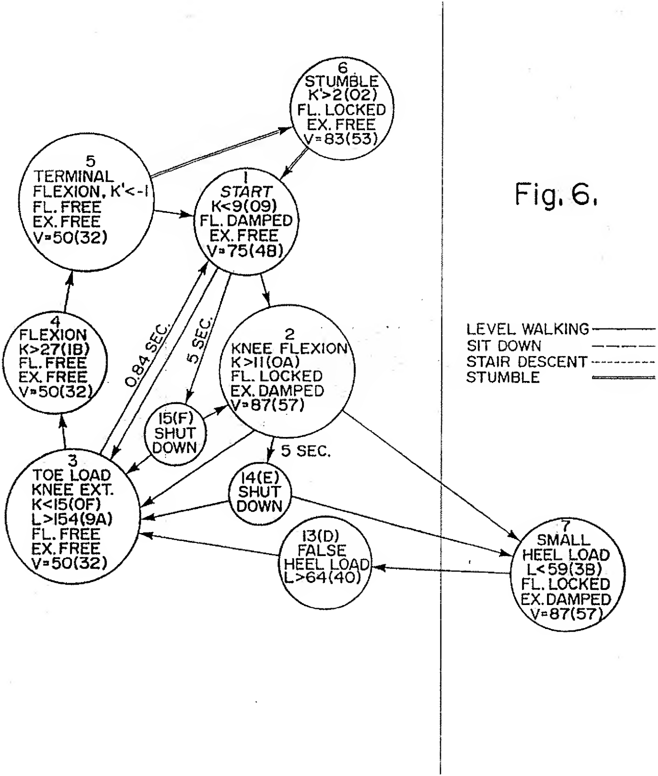

6
STUMBLE
K'>2(02)
FL. LOCKED
EX. FREE
V=83(53)
5
TERMINAL
FLEXION, K'<-1
FL. FREE
EX. FREE
V=50(32)
1
START
K<9(09)
FL. DAMPED
EX. FREE
V=75(48)
4
FLEXION
K>27(IB)
FL. FREE
EX. FREE
V=50(32)
0.84 SEC.
5 SEC.
2
KNEE FLEXION
K>11(OA)
FL. LOCKED
EX. DAMPED
V=87(57)
15(F)
SHUT
DOWN
5 SEC.
3
TOE LOAD
KNEE EXT.
K<15(OF)
L>154(9A)
FL. FREE
EX. FREE
V=50(32)
14(E)
SHUT
DOWN
13(D)
FALSE
HEEL LOAD
L>64(40)
7
SMALL
HEEL LOAD
L<59(3B)
FL. LOCKED
EX. DAMPED
V=87(57)
LEVEL WALKING
SIT DOWN
STAIR DESCENT
STUMBLE

Fig. 6.

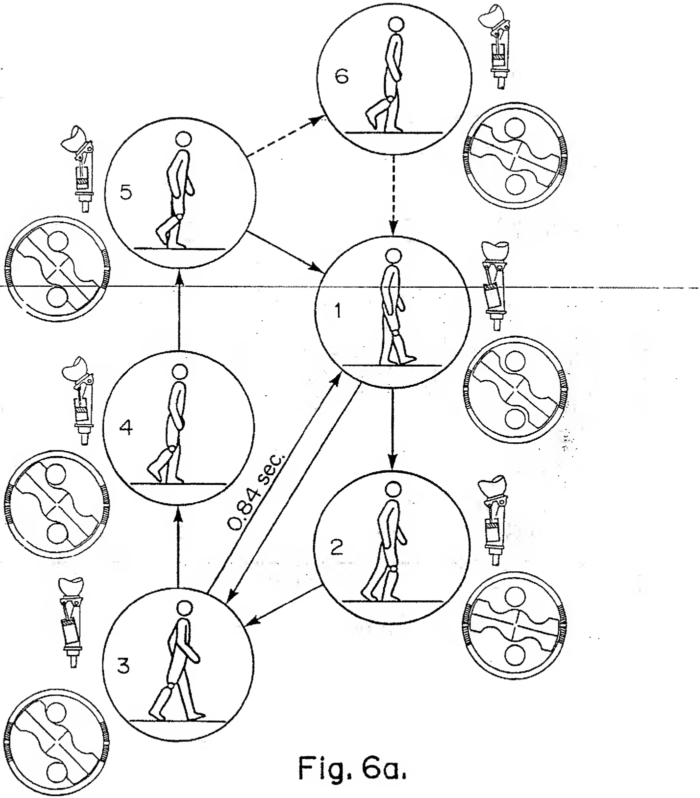


Fig. 6a.

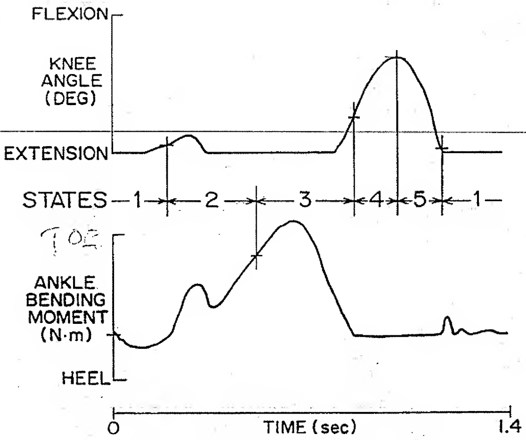

FLEXION
KNEE ANGLE (DEG)
EXTENSION
STATES
1
2
3
4
5
1
TOE
ANKLE BENDING MOMENT (N·m)
HEEL
0
TIME (sec)
1.4

Fig. 7.

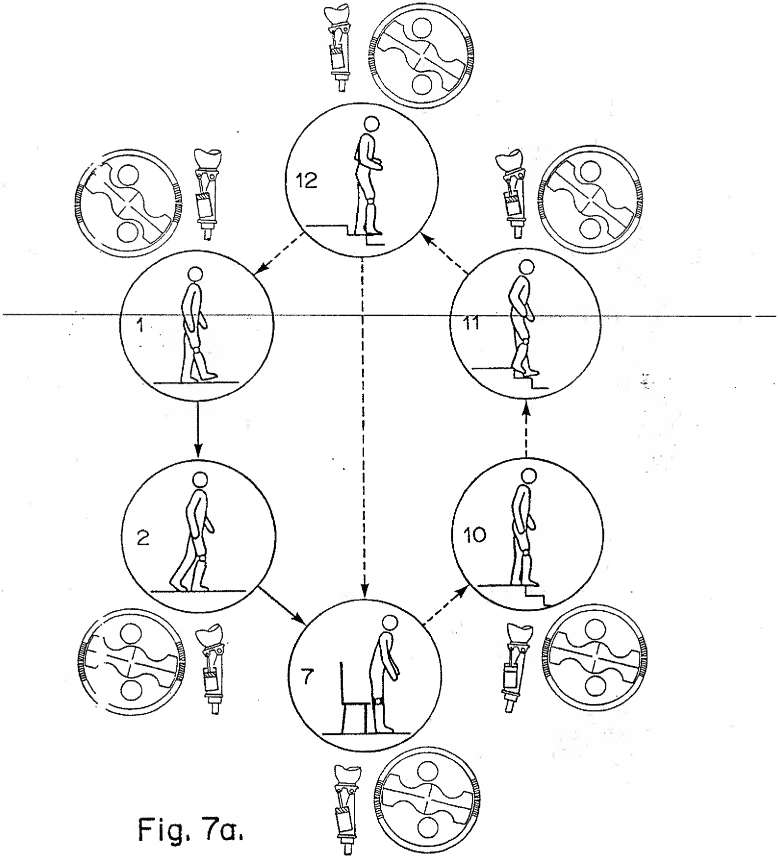
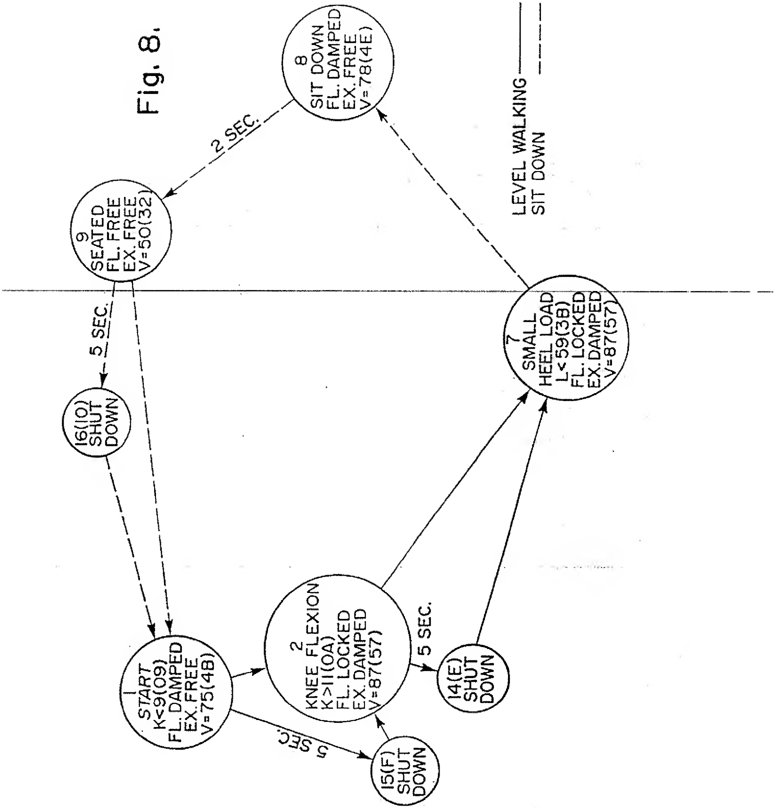


Fig. 7a.

Fig. 8.


8
SIT DOWN
FL. DAMPED
EX. FREE
V=78(4E)
2 SEC.
9
SEATED
FL. FREE
EX. FREE
V=50(32)
5 SEC.
16(10)
SHUT
DOWN
7
SMALL
HEEL LOAD
L<59(3B)
FL. LOCKED
EX.DAMPED
V=87(57)
1
START
K<9(09)
FL.DAMPED
EX.FREE
V=75(4B)
2
KNEE FLEXION
K>11(0A)
FL. LOCKED
EX. DAMPED
V=87(57)
5 SEC.
14(E)
SHUT
DOWN
5 SEC.
15(F)
SHUT
DOWN
LEVEL WALKING
SIT DOWN

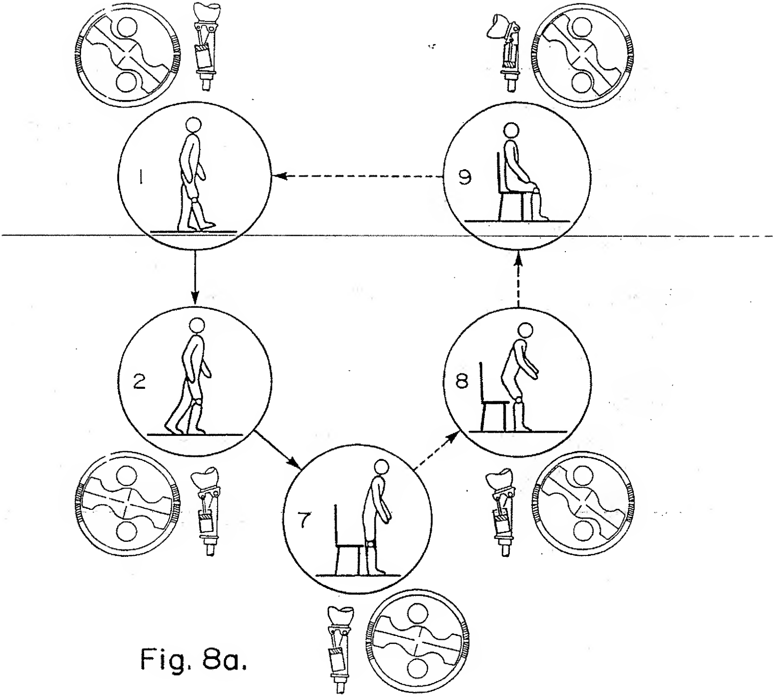


Fig. 8a.

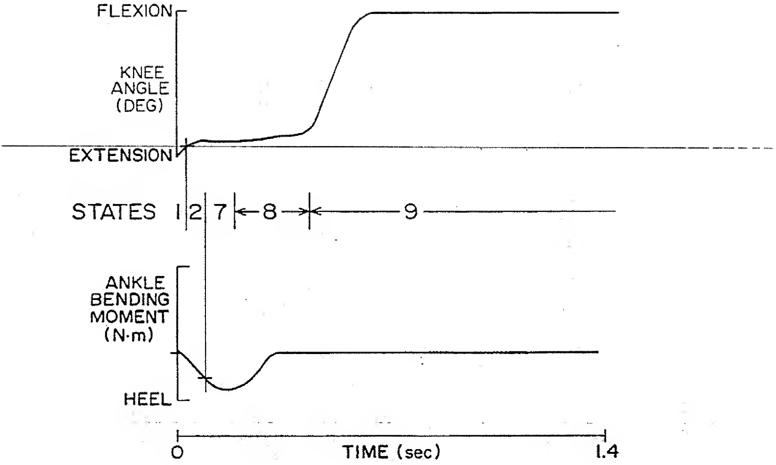
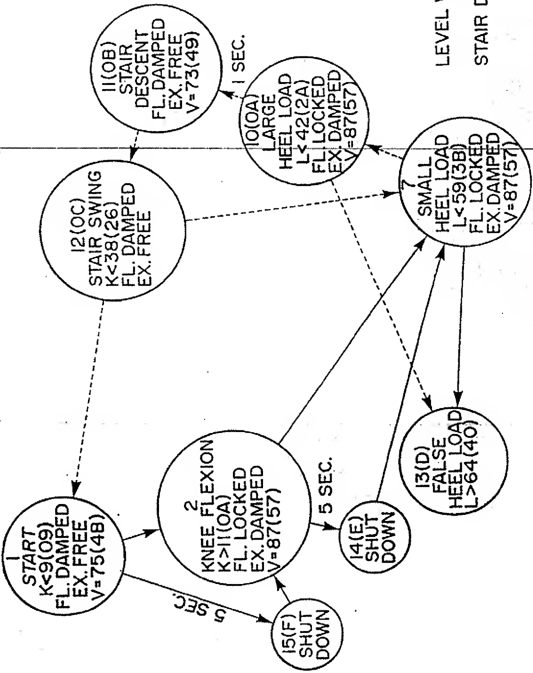

FLEXION
KNEE ANGLE (DEG)
EXTENSION
STATES 1 2 7 8 9
ANKLE BENDING MOMENT (N·m)
HEEL
0
TIME (sec)
1.4

Fig. 9.

Fig. 10.



LEVEL WALKING ——

STAIR DESCENT - - - -

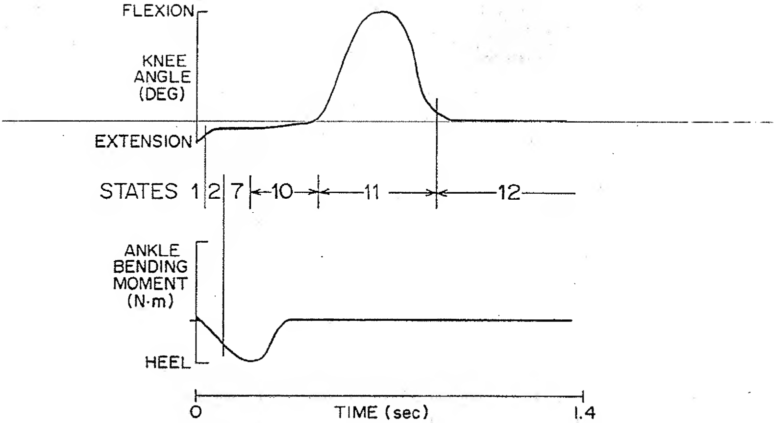

FLEXION
KNEE ANGLE (DEG)
EXTENSION
STATES 1 2 7 10 11 12
ANKLE BENDING MOMENT (N·m)
HEEL
0
TIME (sec)
1.4

Fig. 11.

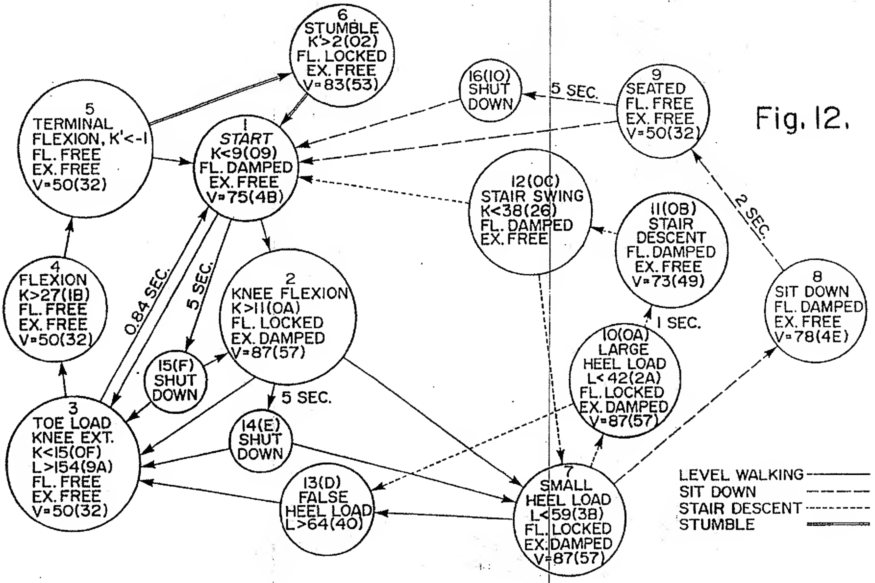

6
STUMBLE
K'>2(02)
FL. LOCKED
EX. FREE
V=83(53)
5
TERMINAL
FLEXION, K'<-I
FL. FREE
EX. FREE
V=50(32)
1
START
K<9(09)
FL. DAMPED
EX. FREE
V=75(4B)
16(10)
SHUT
DOWN
5 SEC.
9
SEATED
FL. FREE
EX. FREE
V=50(32)
12(0C)
STAIR SWING
K<38(26)
FL. DAMPED
EX. FREE
11(0B)
STAIR
DESCENT
FL. DAMPED
EX. FREE
V=73(49)
2 SEC.
8
SIT DOWN
FL. DAMPED
EX. FREE
V=78(4E)
4
FLEXION
K>27(1B)
FL. FREE
EX. FREE
V=50(32)
0.84 SEC.
5 SEC.
2
KNEE FLEXION
K>11(0A)
FL. LOCKED
EX. DAMPED
V=87(57)
15(F)
SHUT
DOWN
5 SEC.
1 SEC.
10(0A)
LARGE
HEEL LOAD
L<42(2A)
FL. LOCKED
EX. DAMPED
V=87(57)
3
TOE LOAD
KNEE EXT.
K<15(0F)
L>154(9A)
FL. FREE
EX. FREE
V=50(32)
14(E)
SHUT
DOWN
13(D)
FALSE
HEEL LOAD
L>64(40)
7
SMALL
HEEL LOAD
L<59(38)
FL. LOCKED
EX. DAMPED
V=87(57)
LEVEL WALKING
SIT DOWN
STAIR DESCENT
STUMBLE

Fig. 12.

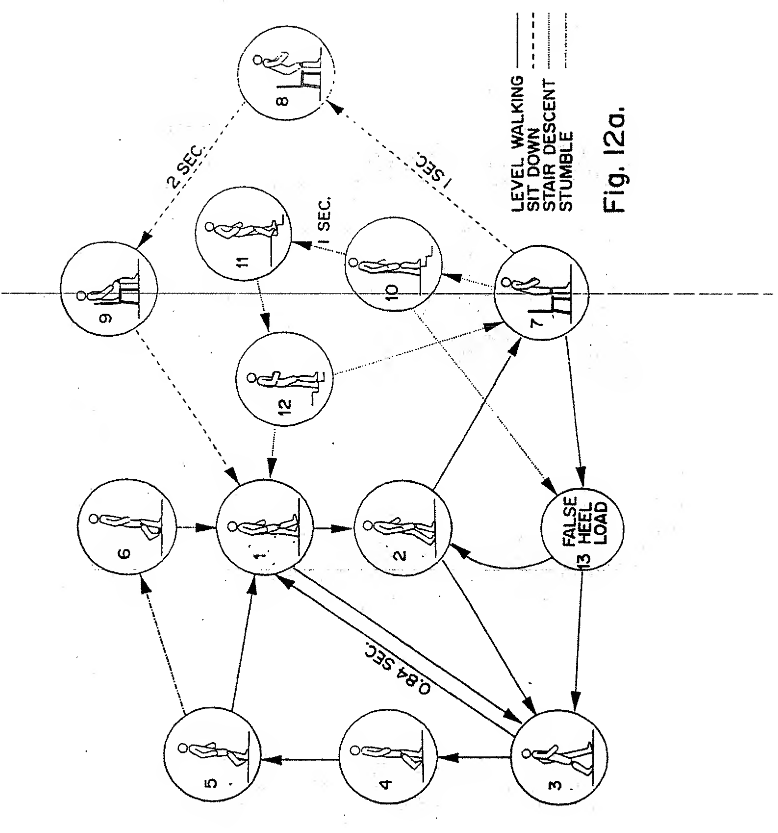

LEVEL WALKING
SIT DOWN
STAIR DESCENT
STUMBLE
8
2 SEC.
I SEC.
I SEC.
9
11
10
7
12
6
1
2
FALSE HEEL LOAD
13
0.84 SEC.
5
4
3

Fig. I2a.

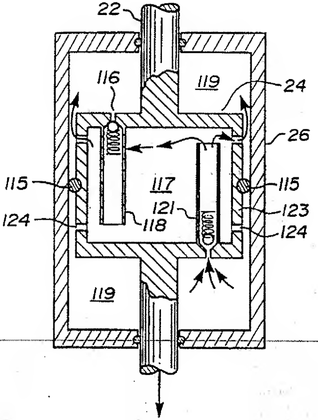

22
116
119
24
26
117
115
115
121
123
124
118
124
119

Fig. 13.

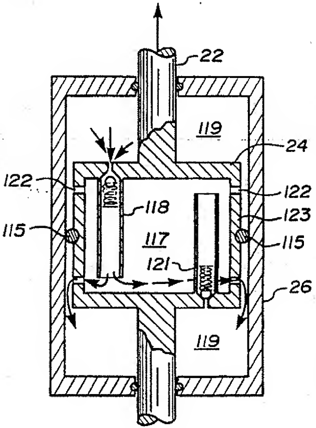

22
119
24
122
122
118
123
115
117
115
121
26
119

Fig. 14.

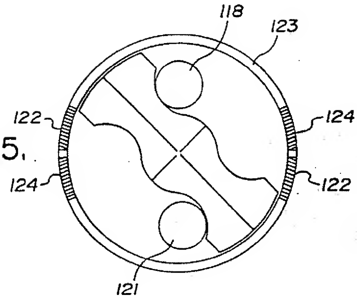


Fig. 15.

Fig. 16.

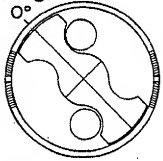


Fig. 17

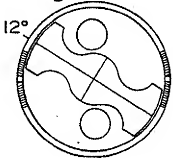


Fig. 18.

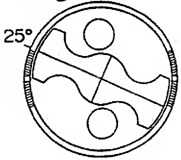


Fig. 19.

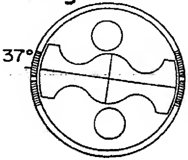


Fig. 20.

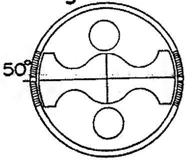


Fig. 21.

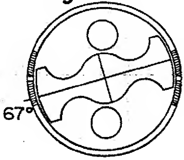


Fig. 22.

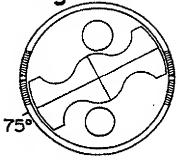


Fig. 23.

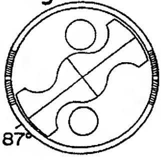
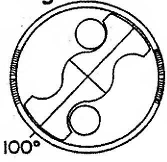


Fig. 24.



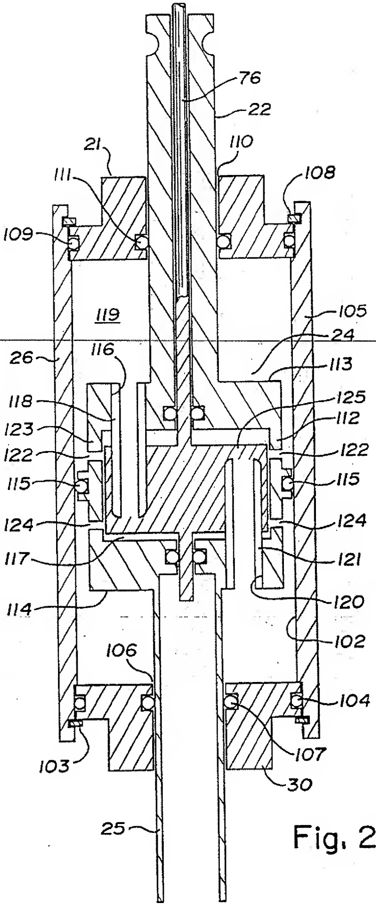

76
22
21
110
111
108
109
119
105
24
26
116
113
118
125
112
123
122
122
115
115
124
124
117
121
120
114
102
106
104
103
107
30
25

Fig. 25.

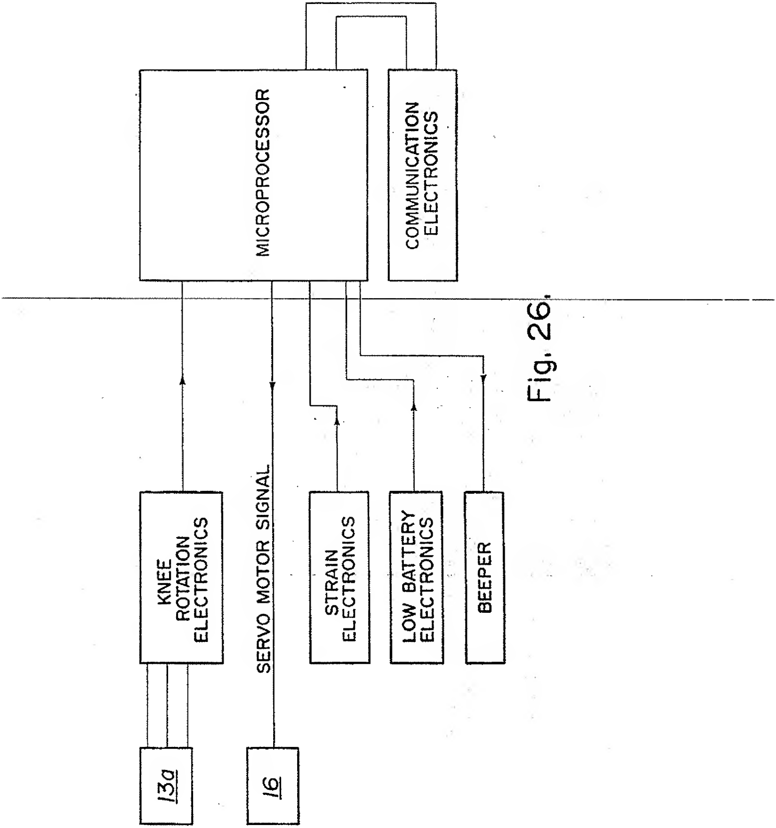

MICROPROCESSOR
COMMUNICATION ELECTRONICS
KNEE ROTATION ELECTRONICS
SERVO MOTOR SIGNAL
STRAIN ELECTRONICS
LOW BATTERY ELECTRONICS
BEEPER
13a
16

Fig. 26.

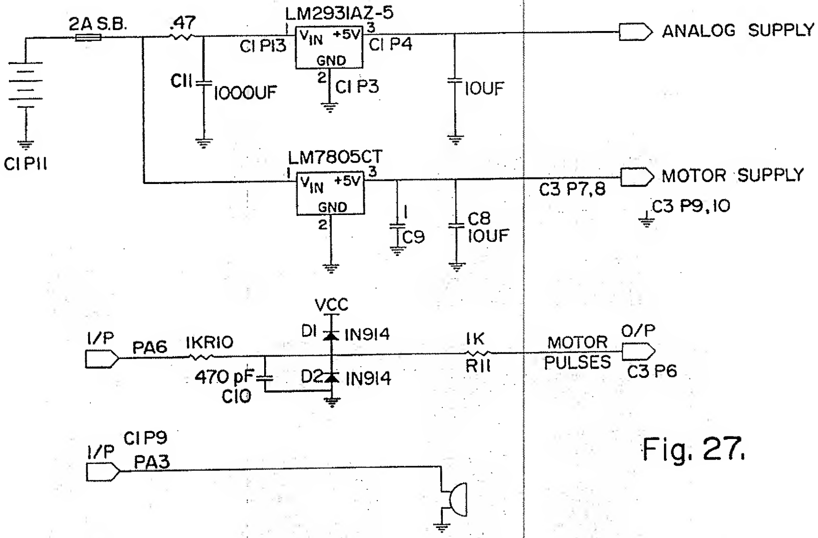

2A S.B.
.47
LM2931AZ-5
V_{IN}
+5V
GND
1
2
3
CI PI3
CI P4
CI P3
CII
IOOOUF
IOUF
ANALOG SUPPLY
CI PII
LM7805CT
V_{IN}
+5V
GND
1
2
3
I
C9
C8
IOUF
C3 P7,8
MOTOR SUPPLY
C3 P9,IO
VCC
DI
IN9I4
D2
IN9I4
I/P
PA6
IKRIO
470 pF
CIO
IK
RII
MOTOR
PULSES
O/P
C3 P6
I/P
CI P9
PA3

Fig. 27.

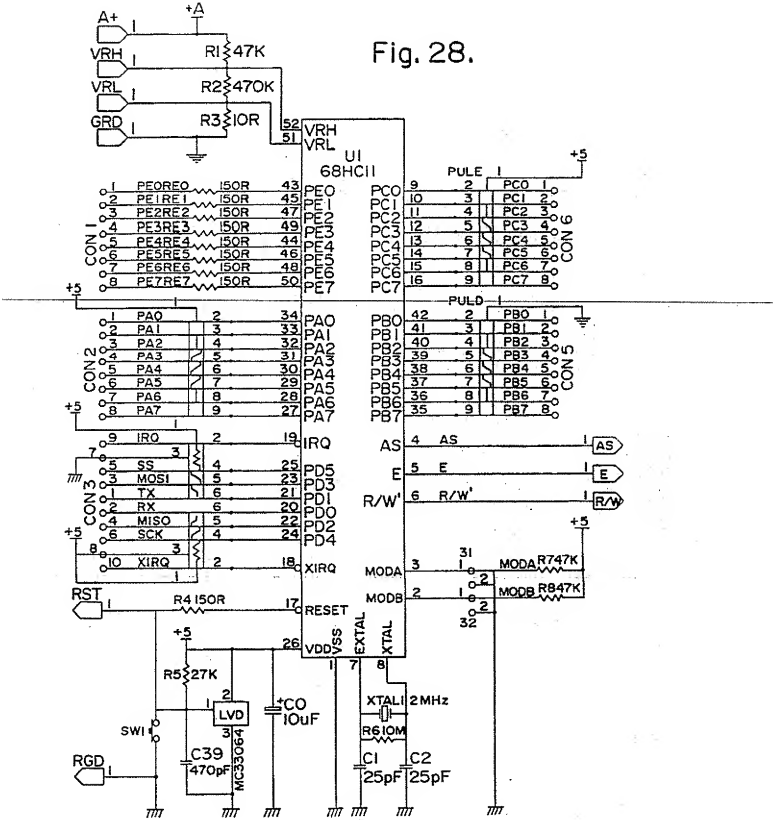
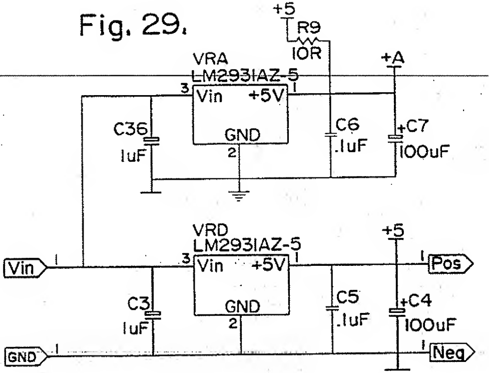


Fig. 28.

A+
+A
VRH
VRL
GRD
R1 47K
R2 470K
R3 10R
52 VRH
51 VRL
U1
68HC11
+5
PULE
CON 1
PE0RE0
PE1RE1
PE2RE2
PE3RE3
PE4RE4
PE5RE5
PE6RE6
PE7RE7
150R
PE0
PE1
PE2
PE3
PE4
PE5
PE6
PE7
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
CON 6
PULD
CON 2
PA0
PA1
PA2
PA3
PA4
PA5
PA6
PA7
PB0
PB1
PB2
PB3
PB4
PB5
PB6
PB7
CON 5
CON3
IRQ
SS
MOSI
TX
RX
MISO
SCK
XIRQ
PD5
PD3
PD1
PD0
PD2
PD4
AS
E
R/W'
MODA
MODB
R7 47K
R8 47K
RESET
RST
R4 150R
VDD
VSS
EXTAL
XTAL
R5 27K
LVD
MC33064
SW1
C39
470pF
CO
10uF
XTAL1 2MHz
R6 10M
C1
25pF
C2
25pF
RGD

Fig. 29.


+5
R9
10R
+A
VRA
LM2931AZ-5
3
Vin
+5V
1
GND
2
C36
1uF
C6
.1uF
+C7
100uF
VRD
LM2931AZ-5
+5
Vin
1
3
Vin
+5V
1
1
Pos
GND
2
C3
1uF
C5
.1uF
+C4
100uF
GND
1
1
Neg

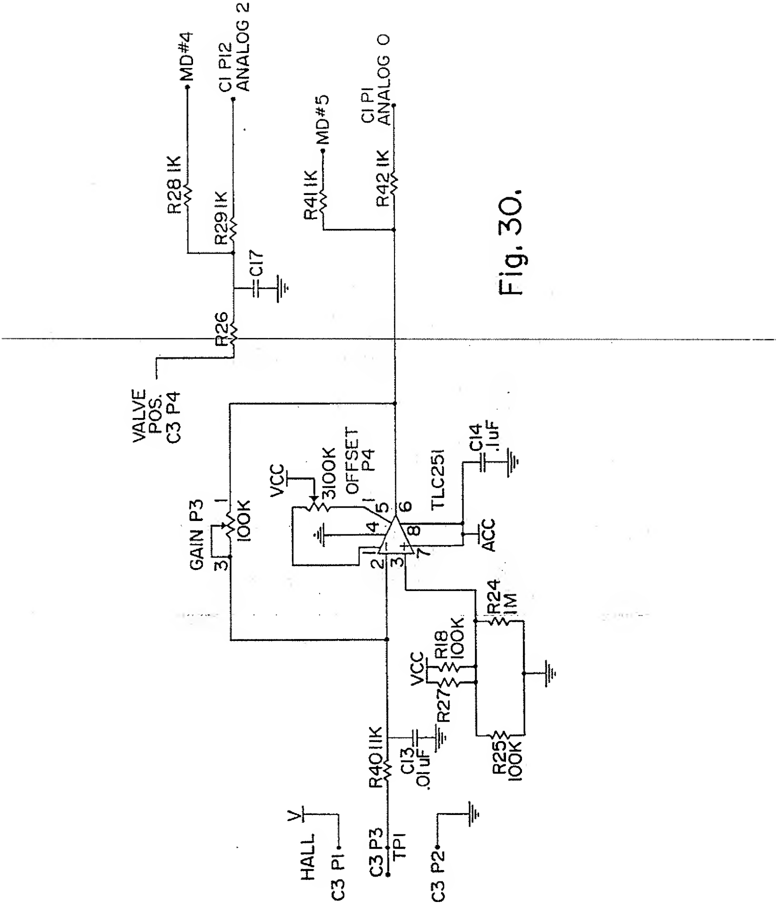

MD#4
C1 P12
ANALOG 2
R28 1K
R29 1K
C17
R26
VALVE
POS.
C3 P4
MD#5
C1 P1
ANALOG 0
R41 1K
R42 1K
GAIN P3
100K
1
3
VCC
3
100K
OFFSET
P4
1
5
6
4
8
2
3
7
TLC251
C14
.1uF
ACC
R24
1M
R18
100K
VCC
R27
R25
100K
R40 11K
C13
.01uF
HALL
C3 P1
V
C3 P3
TP1
C3 P2

Fig. 30.

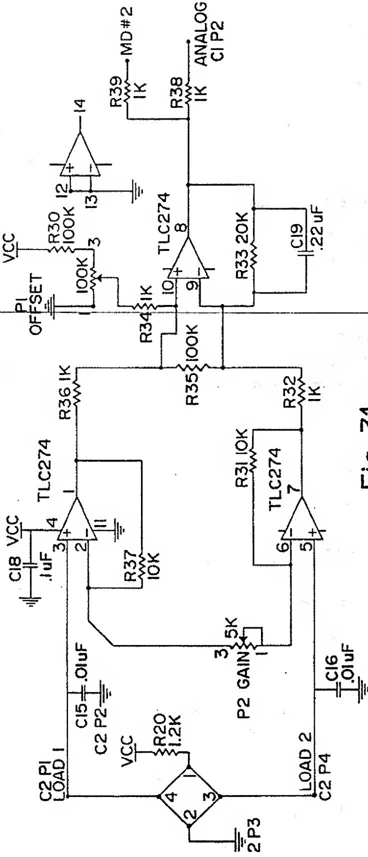

VCC
R30 100K
3
100K
P1 OFFSET
1
R34 1K
12
13
14
MD #2
R39 1K
R38 1K
ANALOG 1 C1 P2
TLC274
8
10
9
R33 20K
C19 .22uF
R35 100K
R36 1K
R32 1K
R31 10K
TLC274
7
6
5
TLC274
1
4
2
3
11
VCC
C18 .1uF
R37 10K
5K
3
1
P2 GAIN
C15 .01uF
C2 P2
C16 .01uF
C2 P1 LOAD 1
VCC
R20 1.2K
4
3
2
1
LOAD 2 C2 P4
2 P3

Fig. 31.

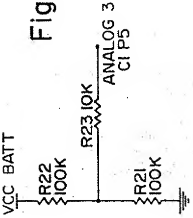
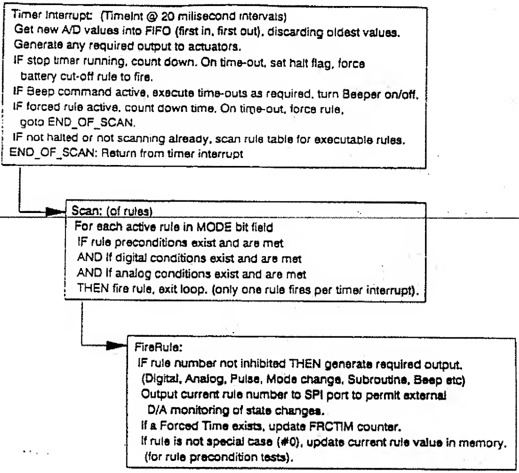

VCC BATT
R22 100K
R23 10K
ANALOG 3 C1 P5
R21 100K

Fig. 32.

Fig. 33.

The core of the program is the Timer Interrupt Service Routine.
Every 20 miliseconds the timer interrupts and...



```
Timer Interrupt: (TimeInt @ 20 milisecond intervals)
 Get new A/D values into FIFO (first in, first out), discarding oldest values.
 Generate any required output to actuators.
 IF stop timer running, count down. On time-out, set halt flag, force
   battery cut-off rule to fire.
 IF Beep command active, execute time-outs as required, turn Beeper on/off.
 IF forced rule active, count down time. On time-out, force rule,
   goto END_OF_SCAN.
 IF not halted or not scanning already, scan rule table for executable rules.
END_OF_SCAN: Return from timer interrupt
```

```
Scan: (of rules)
 For each active rule in MODE bit field
  IF rule preconditions exist and are met
  AND If digital conditions exist and are met
  AND If analog conditions exist and are met
  THEN fire rule, exit loop. (only one rule fires per timer interrupt).
```

```
FireRule:
 IF rule number not inhibited THEN generate required output.
  (Digital, Analog, Pulse, Mode change, Subroutine, Beep etc)
  Output current rule number to SPI port to permit external
    D/A monitoring of state changes.
  If a Forced Time exists, update FRCTIM counter.
  If rule is not special case (#0), update current rule value in memory.
   (for rule precondition tests).
```

A standard implementation of circular buffers is used for interrupt driven input and output via a serial communications port on the microprocessor. On each serial communications interrupt...

```
SERIAL INTERRUPT:
 IF Receiver Interrupt THEN
  WHILE we do not have space in the input circular buffer, WAIT.
  Enque the received character, update in_pointer & counter.
 ELSEIF
  Transmitter interrupt THEN
   IF we have a character to send, send it, update out_pointer & counter.
   ELSE
   turn off transmitter interrupt since nothing left to send.
Return from SERIAL INTERRUPT
```

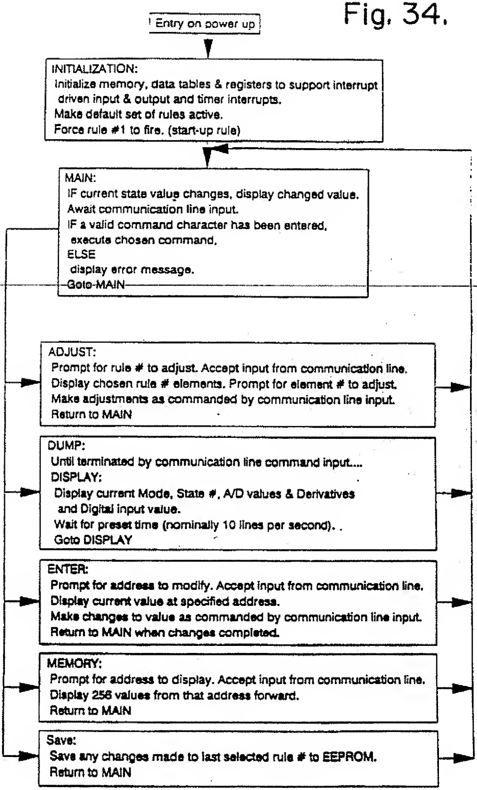


Fig. 34.

Entry on power up

INITIALIZATION:
Initialize memory, data tables & registers to support interrupt driven input & output and timer interrupts.
Make default set of rules active.
Force rule #1 to fire. (start-up rule)

MAIN:
IF current state value changes, display changed value.
Await communication line input.
IF a valid command character has been entered,
execute chosen command,
ELSE
display error message.
Goto MAIN

ADJUST:
Prompt for rule # to adjust. Accept input from communication line.
Display chosen rule # elements. Prompt for element # to adjust.
Make adjustments as commanded by communication line input.
Return to MAIN

DUMP:
Until terminated by communication line command input....
DISPLAY:
Display current Mode, State #, A/D values & Derivatives and Digital input value.
Wait for preset time (nominally 10 lines per second). .
Goto DISPLAY

ENTER:
Prompt for address to modify. Accept input from communication line.
Display current value at specified address.
Make changes to value as commanded by communication line input.
Return to MAIN when changes completed.

MEMORY:
Prompt for address to display. Accept input from communication line.
Display 256 values from that address forward.
Return to MAIN

Save:
Save any changes made to last selected rule # to EEPROM.
Return to MAIN

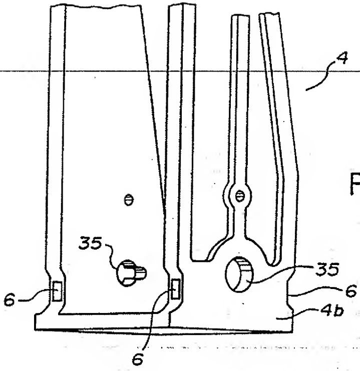


Fig. 35.

(19) Europäisches Patentamt
European Patent Office
Office européen des brevets

(11) Publication number: **0 549 855 A3**

(12) EUROPEAN PATENT APPLICATION

(21) Application number: 92115676.6

(22) Date of filing: 12.09.92

(51) Int. Cl.5: A61F 2/64, A61F 2/68

(30) Priority: 05.12.91 US 804264

(43) Date of publication of application:
07.07.93 Bulletin 93/27

(84) Designated Contracting States:
AT BE CH DE DK ES FR GB GR IE IT LI LU NL PT SE

(88) Date of deferred publication of the search report:
20.10.93 Bulletin 93/42

(71) Applicant: Otto Bock Orthopädische Industrie
Besitz- und
Verwaltungs-Kommanditgesellschaft
Industriestrasse
D-37115 Duderstadt(DE)

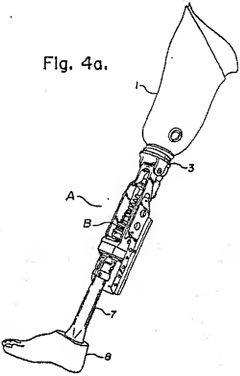
(72) Inventor: James, Kelvin B.
7955 - 98 Avenue
Edmonton, Alberta T6A 0B5(CA)

(74) Representative: Gramm, Werner, Prof.
Dipl.-Ing. et al
Patentanwälte Gramm + Lins
Theodor-Heuss-Strasse 1
D-38122 Braunschweig (DE)

(54) System for controlling artificial knee joint action in an above knee prosthesis.

(57) This invention relates to an above knee prothesis which employs a hydraulic damper to passively regulate the angular velocity or rotation of the artificial knee joint. A programmed microprocessor recognizes common gait patterns from information received from strain and knee angle sensors on the prosthesis. The microprocessor reacts at various transition points in the gait by activating a motor which in turn adjusts a valve assembly in the damper. The valve assembly is capable of variably and separately damping the knee joint motion in each of flexion and extension at the same time. Gait is improved because of the improved extent of control of knee action. In addition, distinct routines such as stair descending and sitting down can also be practised.

Fig. 4a.



Rank Xerox (UK) Business Services

European Patent Office

EUROPEAN SEARCH REPORT

Application Number

EP 92 11 5676

DOCUMENTS CONSIDERED TO BE RELEVANT

Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int. Cl.5)
D,A	FR-A-2 623 086 (ADCRO) * claims 1,2,6; figure 2 * ---	1,2	A 61 F 2/64 A 61 F 2/68
A	GB-A-2 216 426 (KABUSHIKI KAISHA KOBE SEIKO SHO) * page 10, line 3 - line 21; figure 3; page 13, line 7 - page 14, line 25 * ---	1,2,4	
A	INTERNATIONAL JOURNAL OF ELECTRONICS vol. 64, no. 4, 1988, LONDON pages 649 - 656 CHITORE ET AL. 'DIGITAL ELECTRONIC CONTROLLER FOR ABOVE KNEE LEG PROSTHESES' * the whole document * ---	1,2,4 -	
A	GB-A- 826 314 (MAUCH) * claims 1,7; figures 1,2 * ---	1	
A	US-A-2 561 370 (HENSCHKE ET AL.) * claim 1; figures 1,3 * -----	1	

TECHNICAL FIELDS SEARCHED (Int. Cl.5)

A 61 F

The present search report has been drawn up for all claims

Place of search	Date of completion of the search	Examiner
BERLIN	25-03-1993	KANAL P K

CATEGORY OF CITED DOCUMENTS

X : particularly relevant if taken alone
Y : particularly relevant if combined with another document of the same category
A : technological background
O : non-written disclosure
P : intermediate document

T : theory or principle underlying the invention
E : earlier patent document, but published on, or after the filing date
D : document cited in the application
L : document cited for other reasons

& : member of the same patent family, corresponding document

EP92115676

□ CLAIMS INCURRING FEES

The present European patent application comprised at the time of filing more than ten claims.

□ All claims fees have been paid within the prescribed time limit. The present European search report has been drawn up for all claims.

□ Only part of the claims fees have been paid within the prescribed time limit. The present European search report has been drawn up for the first ten claims and for those claims for which claims fees have been paid.

namely claims:

□ No claims fees have been paid within the prescribed time limit. The present European search report has been drawn up for the first ten claims.

☒ LACK OF UNITY OF INVENTION

The Search Division considers that the present European patent application does not comply with the requirement of unity of invention and relates to several inventions or groups of inventions,

namely:

1. Claims 1,2,4-12: Knee prosthesis and method for controlling the knee joint.
2. Claim 3: Hydraulic damper

□ All further search fees have been paid within the fixed time limit. The present European search report has been drawn up for all claims.

□ Only part of the further search fees have been paid within the fixed time limit. The present European search report has been drawn up for those parts of the European patent application which relate to the inventions in respect of which search fees have been paid.

namely claims:

☒ None of the further search fees has been paid within the fixed time limit. The present European search report has been drawn up for those parts of the European patent application which relate to the invention first mentioned in the claims,

namely claims: 1,2,4-12